US006005942A

United States Patent [19]

Chan et al.

[11] **Patent Number: 6,005,942**

[45] **Date of Patent: Dec. 21, 1999**

[54] **SYSTEM AND METHOD FOR A MULTI-APPLICATION SMART CARD WHICH CAN FACILITATE A POST-ISSUANCE DOWNLOAD OF AN APPLICATION ONTO THE SMART CARD**

[75] Inventors: **Alfred Chan**, Daly City; **Marc B. Kekicheff**, Palo Alto; **Joel M. Weise**, Burlingame; **David C. Wentker**, San Francisco, all of Calif.

[73] Assignee: **Visa International Service Association**, Foster City, Calif.

[21] Appl. No.: **09/046,993**

[22] Filed: **Mar. 24, 1998**

Related U.S. Application Data

[60] Provisional application No. 60/061,763, Oct. 14, 1997, and provisional application No. 60/041,468, Mar. 24, 1997.

[51] **Int. Cl.**[6] **H04L 9/00**; H04L 9/08; G07F 7/08

[52] **U.S. Cl.** **380/25**; 380/9; 380/21; 380/23; 380/24; 380/29; 380/30; 380/49; 380/50; 235/379; 235/380

[58] **Field of Search** 380/4, 23, 24, 380/25, 49, 50, 59, 9, 21, 29, 30; 235/379, 380, 382; 379/93.01, 93.05, 93.06, 93.12

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,742,215	5/1988	Daughters et al.	235/487
5,332,889	7/1994	Lundstrom et al.	235/380
5,378,884	1/1995	Lundstrom et al.	235/380 X
5,530,232	6/1996	Taylor	235/380
5,583,933	12/1996	Mark	380/9 X

FOREIGN PATENT DOCUMENTS

E 100227	11/1994	Austria .
0193635 A1	9/1986	European Pat. Off. .
19607363 A1	9/1996	Germany .

OTHER PUBLICATIONS

EPO, International Search Report, Jul. 3, 1998, International Application No. PCT/US 98/05674.

Carol Hovenga Fancher,"In Your Pocket Smart Cards", Feb. 1997, IEEE.

Chaum et al., "Smart Card 2000: The Future of IC Cards", Oct. 19, 1987, Elsevier Science Publishers B.V.

Steven Levy, "E–Money (That's What I Want)", Dec. 1994, Wired Magazine.

Carol H. Fancher, "Smart Cards as Potential Applications Grow, Computers in the Wallet are Making Unobtrusive Inroads", Aug. 1996, Scientific American Website.

Jerome Svigals, "Smart Cards The New Bank Cards", 1985, MacMillan Publishing Company.

Roy Bright, "Smart Cards: Principles, Practice, Applications", 1988, Ellis Horwood Limited.

Jerome Svigals, "Smart Cards The Ultimate Personal Computer", 1985, MacMillan Publishing Company.

Hawkes et al., "Integrated Circuit Cards, Tags and Tokens", 1990, BSP Professional Books.

Hiro Shogase, "The Very Smart Card: A Plastic Packet Bank", Oct. 1988, IEEE Spectrum.

David Naccache, "Cryptographic Smart Cards", Jun. 3, 1996, IEEE Micro 1996 Website.

Zoreda et al., "Smart Cards", 1994, Artech House.

"Identification Card Systems—Inter–Sector Electronic Purse Part 1: Concepts and Structures", 1994, European Standard, prEN 1546.

"Identification Card Systems—Inter–Sector Electronic Purse Part 2: Security Architecture", 1994, European Standard, prEN XXXXX–2.

"Identification Card Systems—Inter–Sector Electronic Purse Part 3: Data Elements and Interchanges", 1994, European Prestandard, prEN 1546–3.

"Identification Card Systems—Inter–Sector Electronic Purse Part 4: Devices", 1994, European Prestandard, prEN 1546–4.

"Identification Cards—Integrated Circuit(s) Cards With Contacts Part 1: Physical Characteristics", 1987, International Standard, ISO 7816–1, First Edition.

"Identification Cards—Integrated Circuit(s) Cards With Contacts Part 2: Dimensions and Location of the Contacts", 1988, International Standard, ISO 7816–2, First Edition.

"Identification Cards—Integrated Circuit(s) Cards With Contacts Part 3: Electronic Signals and Transmission Protocols", International Standard, ISO/IEC 7816–3, First Edition.

"Identification Cards—Integrated Circuit(s) Cards With Contacts Part 4: Inter–Industry Commands for Interchange", International Standard, ISO/IEC 7816–4, First Edition.

"Identification Cards—Integrated Circuit(s) Cards With Contacts Part 5: Numbering System and Registration Procedure for Application Identifiers", 1993, International Standard, ISO/IEC DIS 7816–5.

"Identification Cards—Physical Characteristics", 1995, International Standard, ISO/IEC 7810, Second Edition.

"Identification Cards–Recording Technique—Part 1: Embossing", 1995, International Standard, ISO/IEC 7811–1, Second Edition.

(List continued on next page.)

Primary Examiner—Bernarr E. Gregory
Attorney, Agent, or Firm—Beyer & Weaver, LLP

[57] **ABSTRACT**

A system and method allow card issuers to securely add applications during the lifetime of the card after the card has already been issued (post issuance). Loading of an application and/or objects from an application server via a card acceptance device (and its supporting system infrastructure delivery mechanism) onto a card post issuance is performed in a secure and confidential manner. A smart card includes a card domain application that manages the card. Any number of security domain applications on the card provide security for loaded applications by managing keys; each application is associated with a security domain. Each of the card domain and security domains has a command interface for off-card communication, and an API for internal card use. The card life cycle includes the states of masked, initialized, load secured and blocked. An application life cycle includes the states of not available, loaded, installed, registered, personalized, activated and blocked. An application can block the card.

24 Claims, 15 Drawing Sheets

OTHER PUBLICATIONS

"Identification Cards–Recording Technique–Part 2: Magnetic Stripe", 1995, International Standard, ISO/IEC 7811–2, Second Edition.

"Identification Cards–Recording Technique–Part 3: Location of Embossed Characters on ID–1 Cards", 1995, International Standard, ISO/IEC 7811–4, Second Edition.

"Identification Cards–Recording Technique–Part 5: Location of Read–Write Magnetic Track—Track 3", 1995, International Standard, ISO/IEC 7811–5, Second Edition.

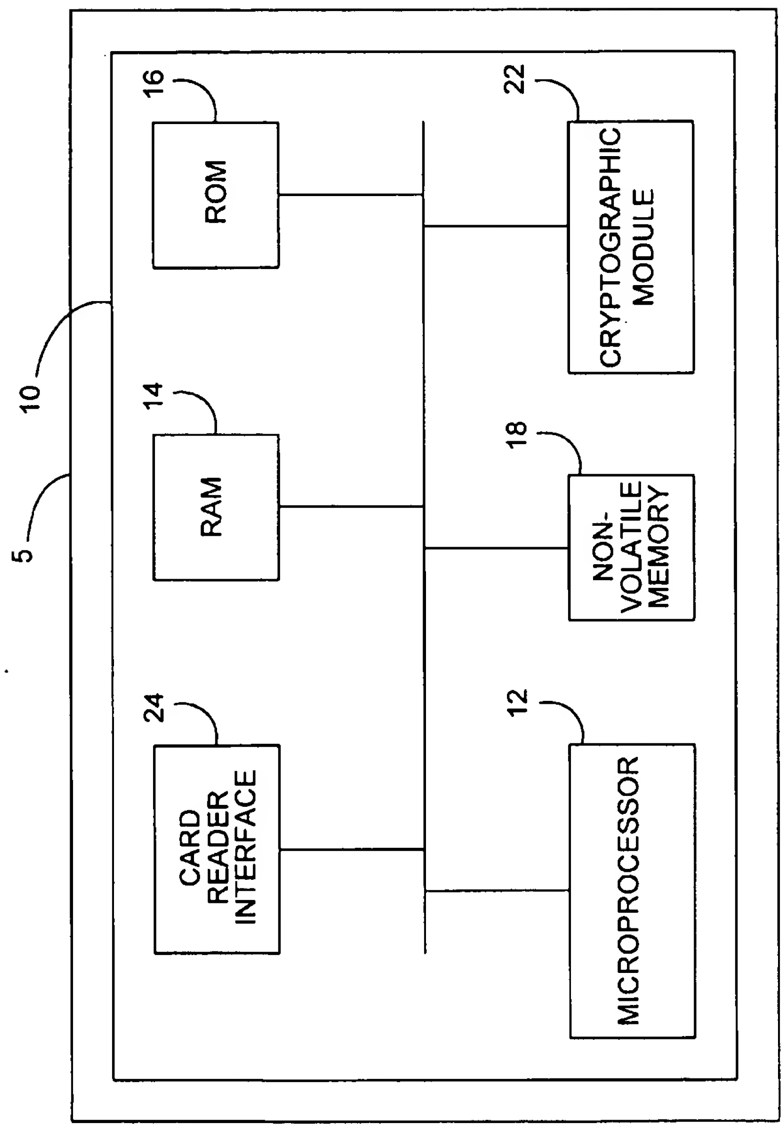
"Identification Cards–Recording Technique–Part 6: Magnetic Stripe–High Coercivity", 1996, International Standard, ISO/IEC 7811–6, First Edition.

"Identification Cards–Financial Transaction Cards", 1990, International Standard, ISO/IEC 7813, Fourth Edition.

"Identification Cards—Financial Transaction Cards Amendment 1", 1996, International Standard, ISO/IEC 7813, Fourth Edition.

"Identification Cards—Countless Integrated Circuit(s) Cards—Part 1: Physical Characteristics", 1992, International Standard, ISO/IEC 10536–1, First Edition.

"Identification Cards—Contactless Integrated Circuit(s) Cards—Part 2: Dimensions and Location of Coupling Areas", 1995, International Standard, ISO/IEC 10536–2, First Edition.



SMART CARD

FIG. 1
(PRIOR ART)

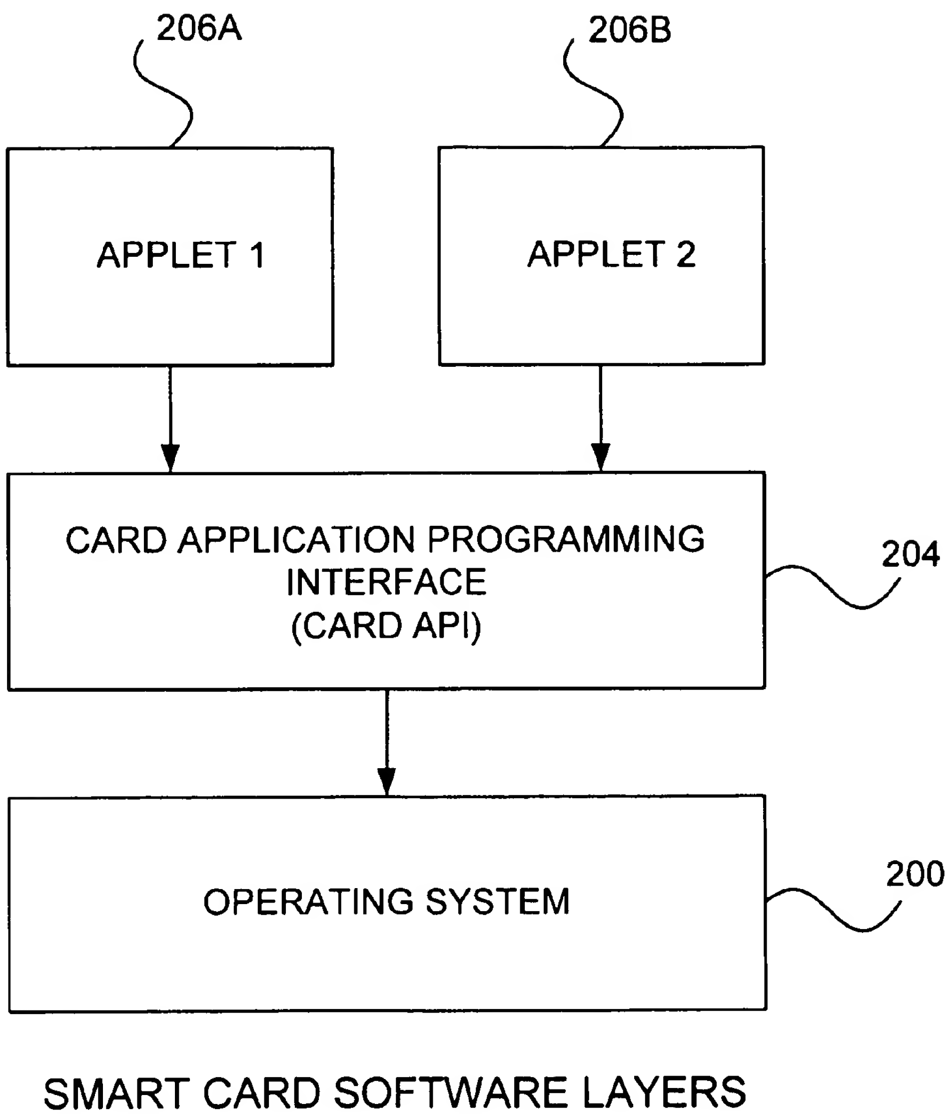


FIG. 2
(PRIOR ART)

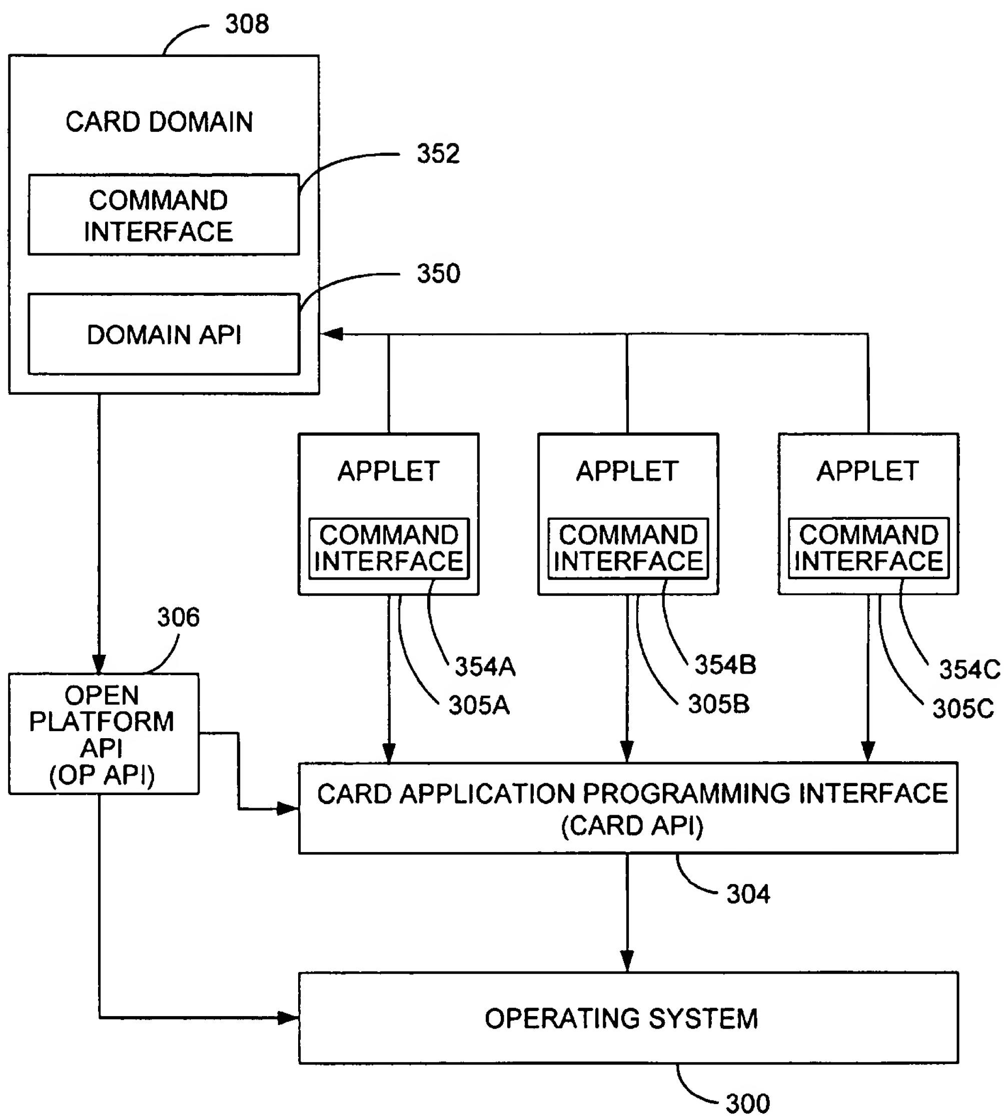

308
CARD DOMAIN
COMMAND INTERFACE
352
DOMAIN API
350
APPLET
COMMAND INTERFACE
APPLET
COMMAND INTERFACE
APPLET
COMMAND INTERFACE
306
OPEN PLATFORM API (OP API)
354A
305A
354B
305B
354C
305C
CARD APPLICATION PROGRAMMING INTERFACE (CARD API)
304
OPERATING SYSTEM
300

FIG. 3A

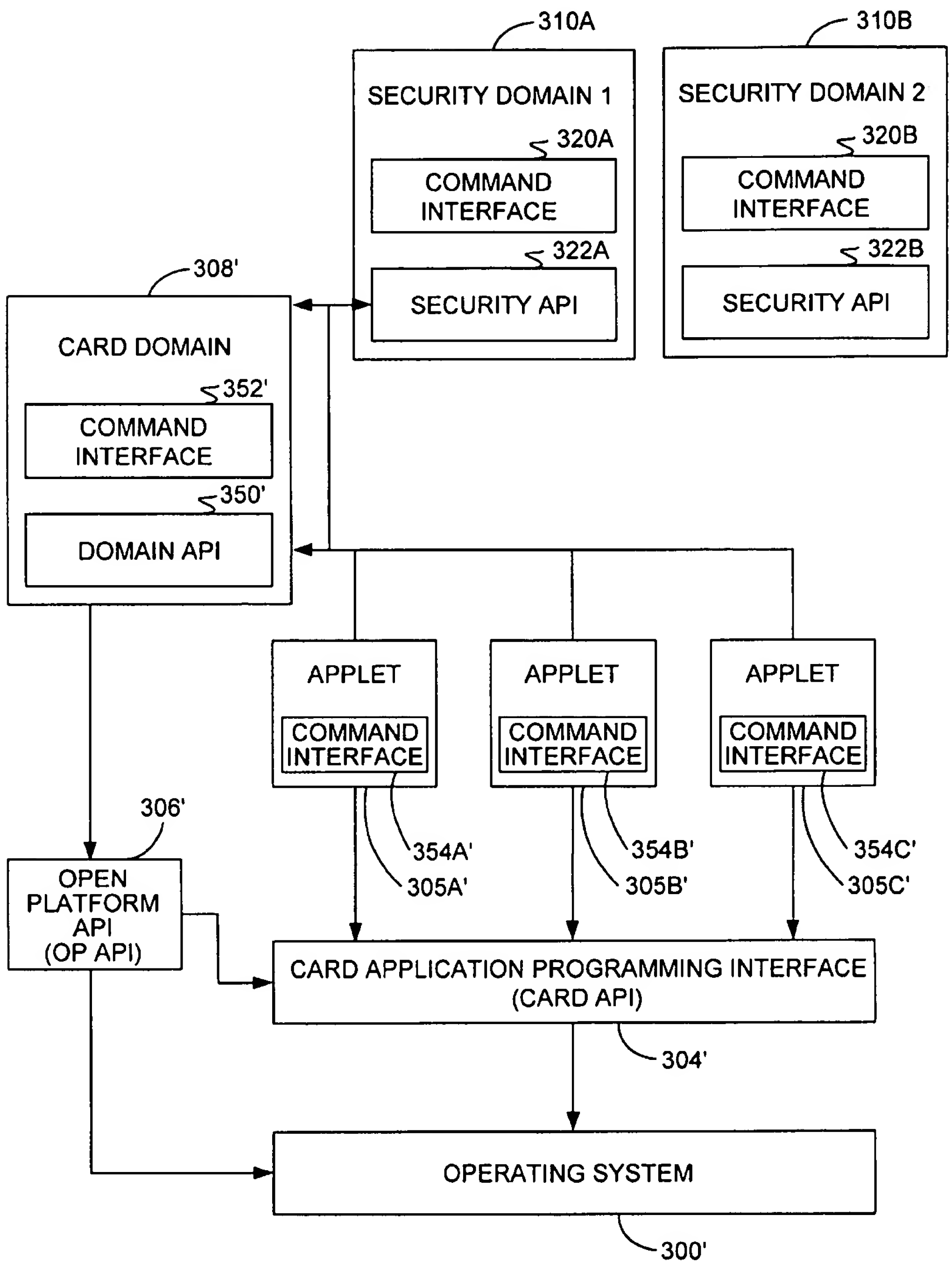

310A
SECURITY DOMAIN 1
320A
COMMAND INTERFACE
322A
SECURITY API
310B
SECURITY DOMAIN 2
320B
COMMAND INTERFACE
322B
SECURITY API
308'
CARD DOMAIN
352'
COMMAND INTERFACE
350'
DOMAIN API
APPLET
COMMAND INTERFACE
APPLET
COMMAND INTERFACE
APPLET
COMMAND INTERFACE
306'
OPEN PLATFORM API (OP API)
354A'
305A'
354B'
305B'
354C'
305C'
CARD APPLICATION PROGRAMMING INTERFACE (CARD API)
304'
OPERATING SYSTEM
300'

FIG. 3B

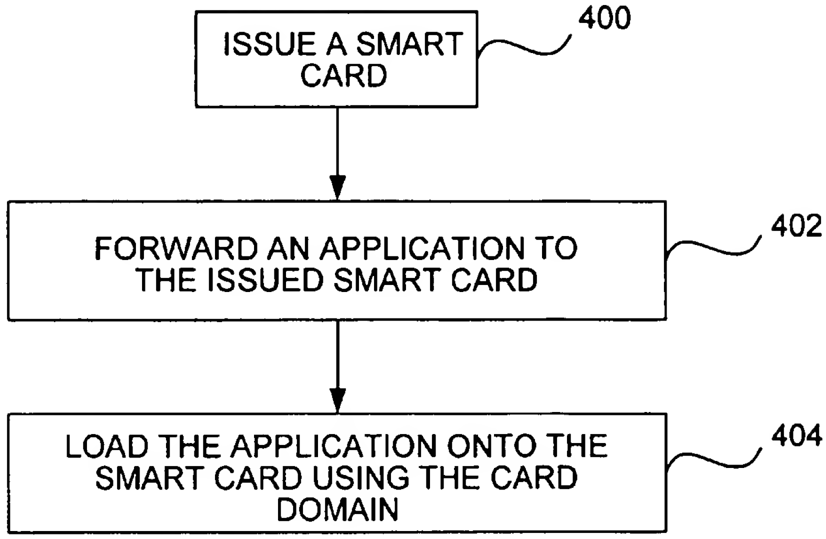

ISSUE A SMART CARD
400
FORWARD AN APPLICATION TO THE ISSUED SMART CARD
402
LOAD THE APPLICATION ONTO THE SMART CARD USING THE CARD DOMAIN
404

FIG. 4

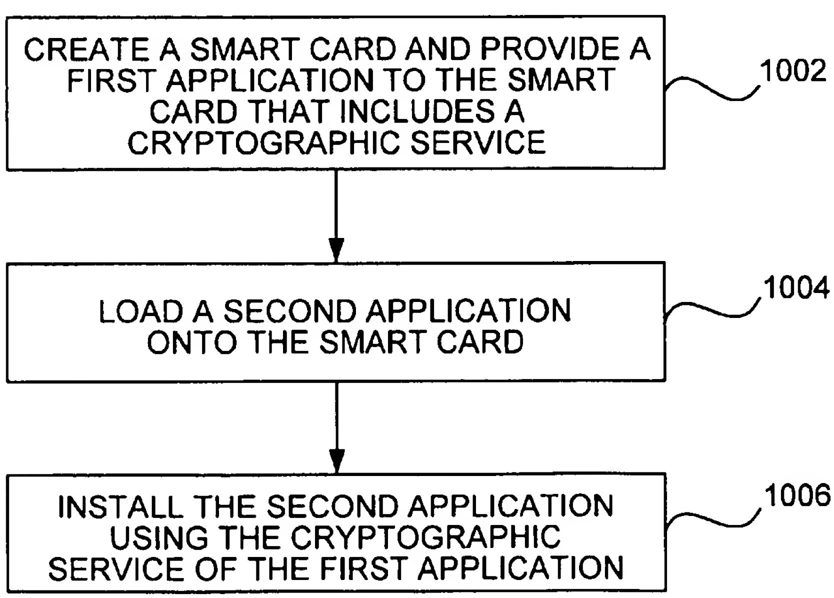

CREATE A SMART CARD AND PROVIDE A FIRST APPLICATION TO THE SMART CARD THAT INCLUDES A CRYPTOGRAPHIC SERVICE
1002
LOAD A SECOND APPLICATION ONTO THE SMART CARD
1004
INSTALL THE SECOND APPLICATION USING THE CRYPTOGRAPHIC SERVICE OF THE FIRST APPLICATION
1006

FIG. 5

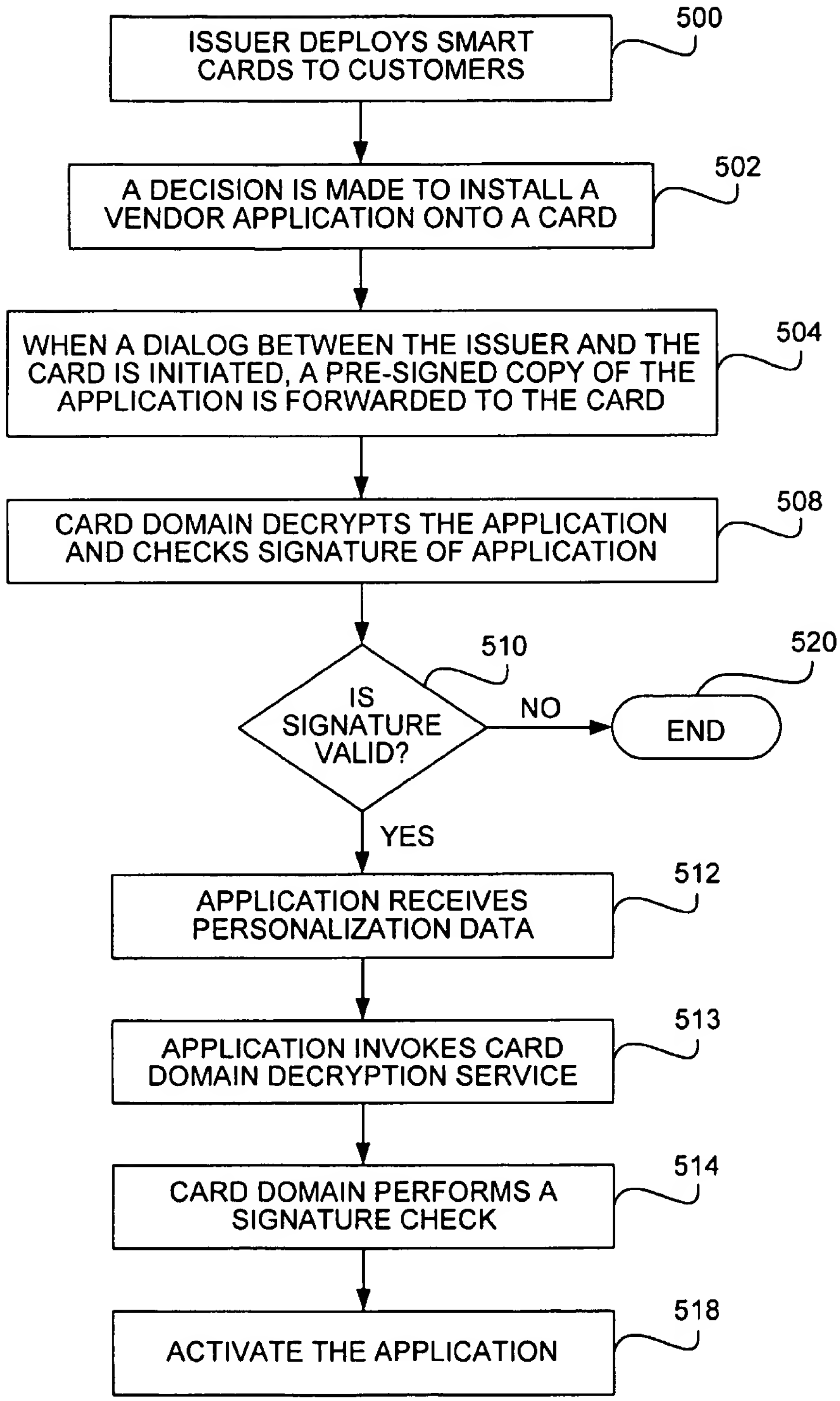

ISSUER DEPLOYS SMART CARDS TO CUSTOMERS
500
A DECISION IS MADE TO INSTALL A VENDOR APPLICATION ONTO A CARD
502
WHEN A DIALOG BETWEEN THE ISSUER AND THE CARD IS INITIATED, A PRE-SIGNED COPY OF THE APPLICATION IS FORWARDED TO THE CARD
504
CARD DOMAIN DECRYPTS THE APPLICATION AND CHECKS SIGNATURE OF APPLICATION
508
IS SIGNATURE VALID?
510
NO
END
520
YES
APPLICATION RECEIVES PERSONALIZATION DATA
512
APPLICATION INVOKES CARD DOMAIN DECRYPTION SERVICE
513
CARD DOMAIN PERFORMS A SIGNATURE CHECK
514
ACTIVATE THE APPLICATION
518

FIG. 6

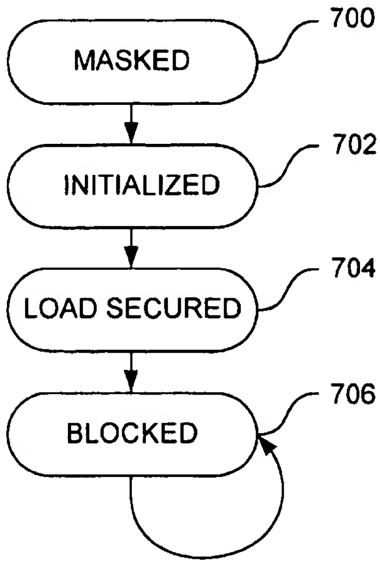

MASKED
700
INITIALIZED
702
LOAD SECURED
704
BLOCKED
706

FIG. 7A

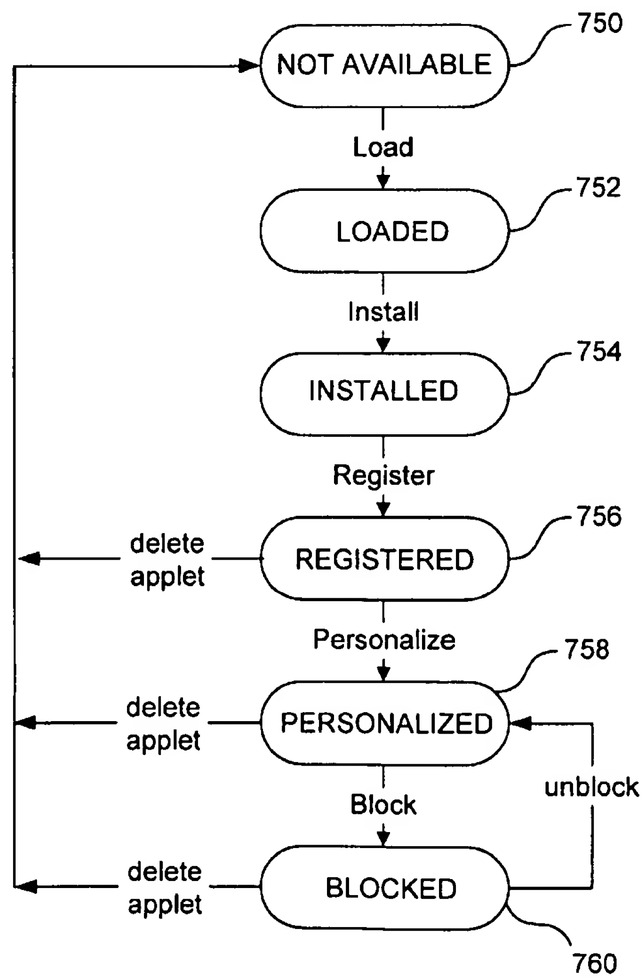

NOT AVAILABLE
750
Load
LOADED
752
Install
INSTALLED
754
Register
REGISTERED
756
delete applet
Personalize
PERSONALIZED
758
delete applet
Block
unblock
BLOCKED
760
delete applet

FIG. 7B

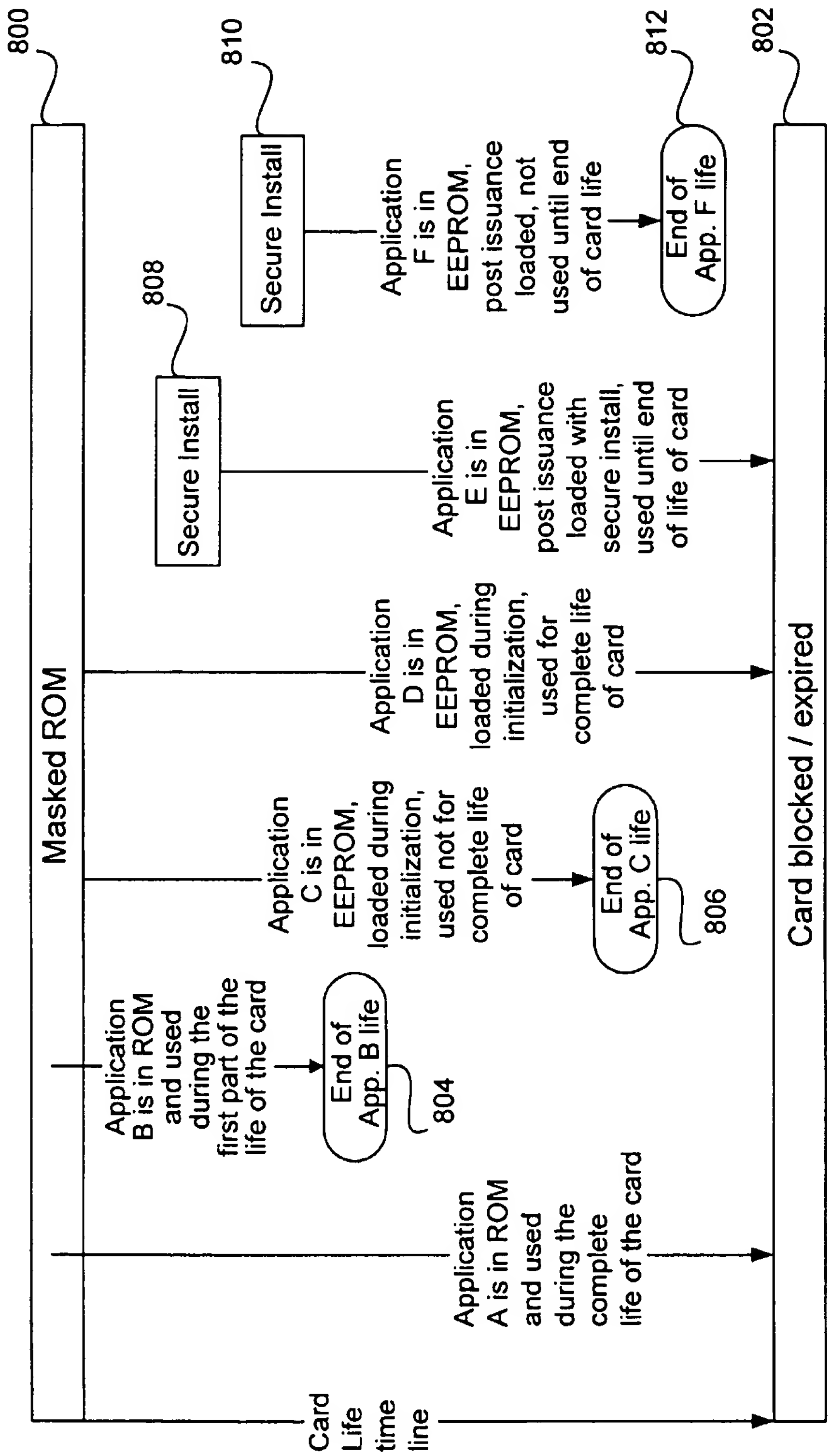

800
Masked ROM
810
Secure Install
808
Secure Install
812
802
Card blocked / expired
Card Life time line
Application A is in ROM and used during the complete life of the card
Application B is in ROM and used during the first part of the life of the card
End of App. B life
804
Application C is in EEPROM, loaded during initialization, used not for complete life of card
End of App. C life
806
Application D is in EEPROM, loaded during initialization, used for complete life of card
Application E is in EEPROM, post issuance loaded with secure install, used until end of life of card
Application F is in EEPROM, post issuance loaded, not used until end of card life
End of App. F life

FIG. 8

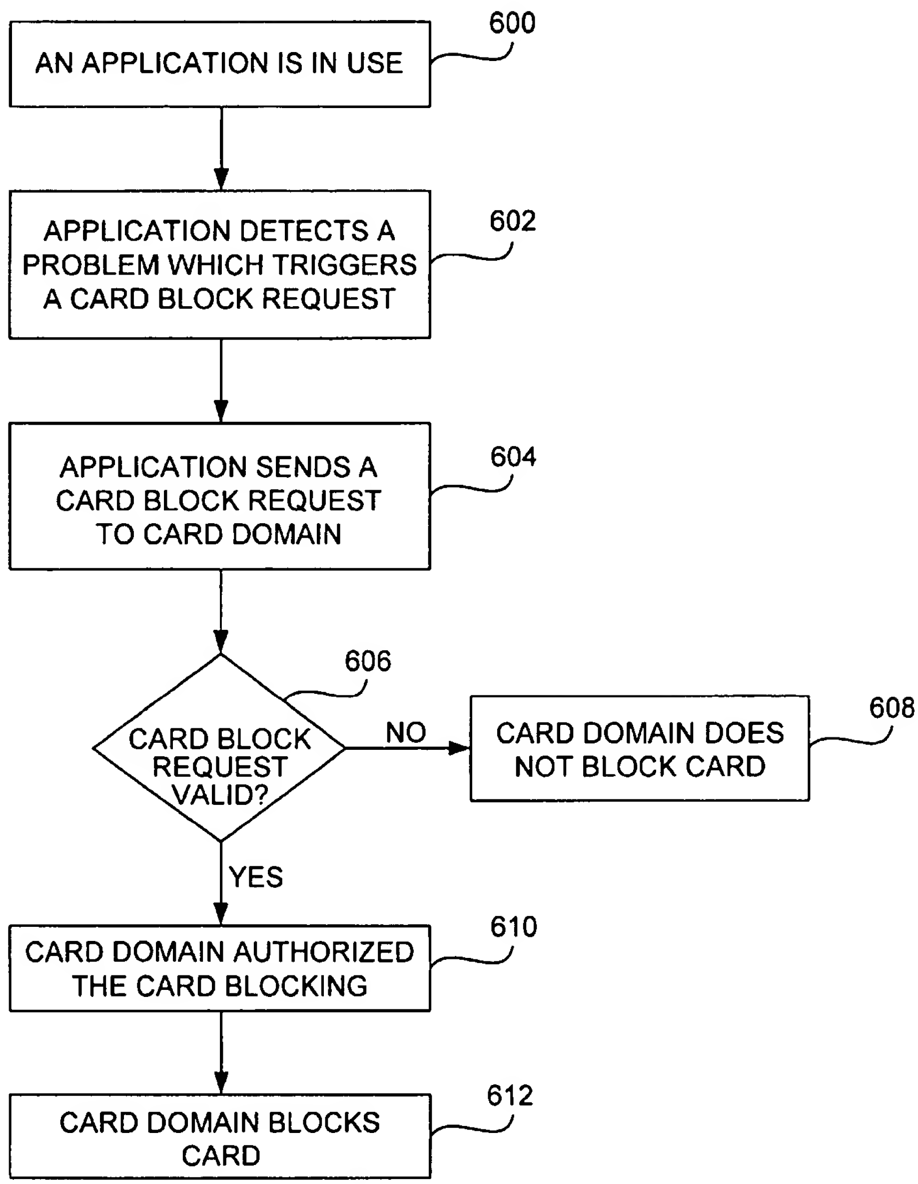

AN APPLICATION IS IN USE
600
APPLICATION DETECTS A PROBLEM WHICH TRIGGERS A CARD BLOCK REQUEST
602
APPLICATION SENDS A CARD BLOCK REQUEST TO CARD DOMAIN
604
CARD BLOCK REQUEST VALID?
606
NO
CARD DOMAIN DOES NOT BLOCK CARD
608
YES
CARD DOMAIN AUTHORIZED THE CARD BLOCKING
610
CARD DOMAIN BLOCKS CARD
612

FIG. 9

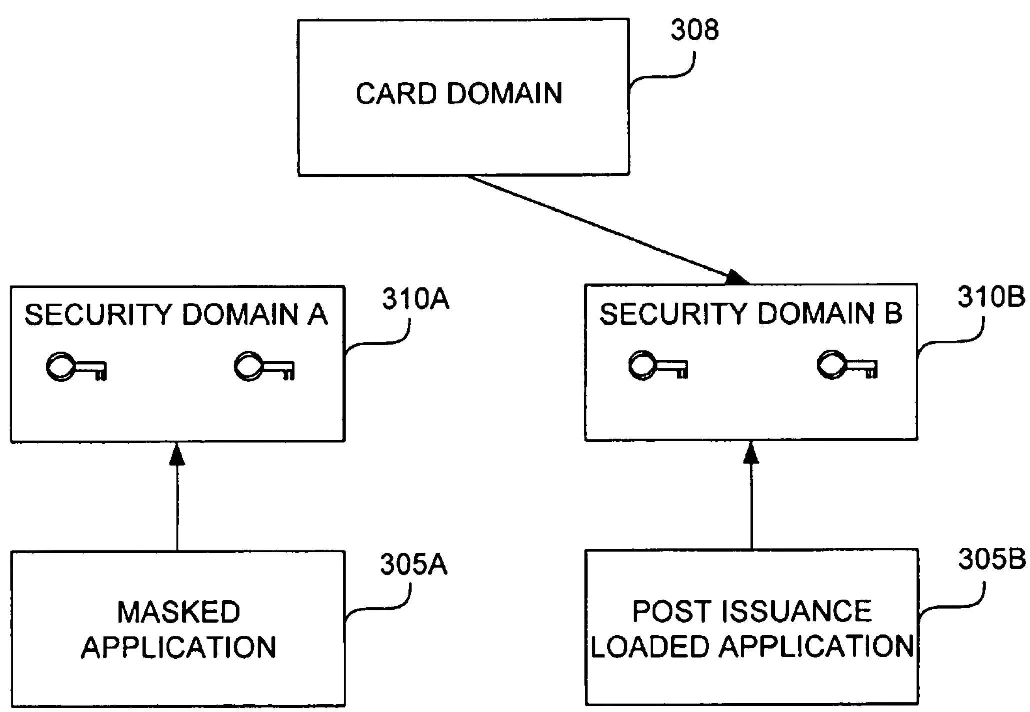

308
CARD DOMAIN
SECURITY DOMAIN A
310A
SECURITY DOMAIN B
310B
MASKED APPLICATION
305A
POST ISSUANCE LOADED APPLICATION
305B

FIG. 10

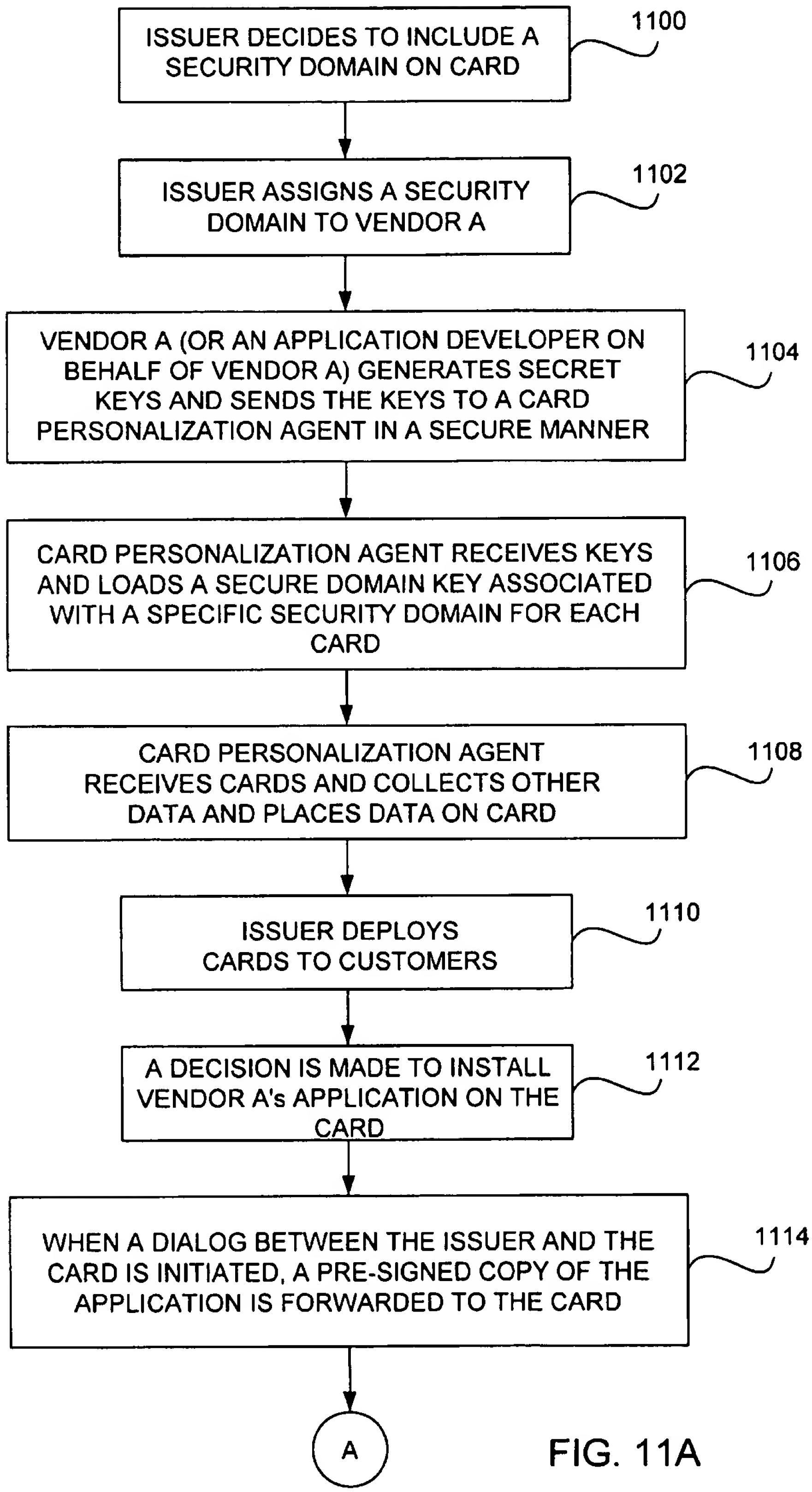

ISSUER DECIDES TO INCLUDE A SECURITY DOMAIN ON CARD
1100
ISSUER ASSIGNS A SECURITY DOMAIN TO VENDOR A
1102
VENDOR A (OR AN APPLICATION DEVELOPER ON BEHALF OF VENDOR A) GENERATES SECRET KEYS AND SENDS THE KEYS TO A CARD PERSONALIZATION AGENT IN A SECURE MANNER
1104
CARD PERSONALIZATION AGENT RECEIVES KEYS AND LOADS A SECURE DOMAIN KEY ASSOCIATED WITH A SPECIFIC SECURITY DOMAIN FOR EACH CARD
1106
CARD PERSONALIZATION AGENT RECEIVES CARDS AND COLLECTS OTHER DATA AND PLACES DATA ON CARD
1108
ISSUER DEPLOYS CARDS TO CUSTOMERS
1110
A DECISION IS MADE TO INSTALL VENDOR A's APPLICATION ON THE CARD
1112
WHEN A DIALOG BETWEEN THE ISSUER AND THE CARD IS INITIATED, A PRE-SIGNED COPY OF THE APPLICATION IS FORWARDED TO THE CARD
1114
A

FIG. 11A

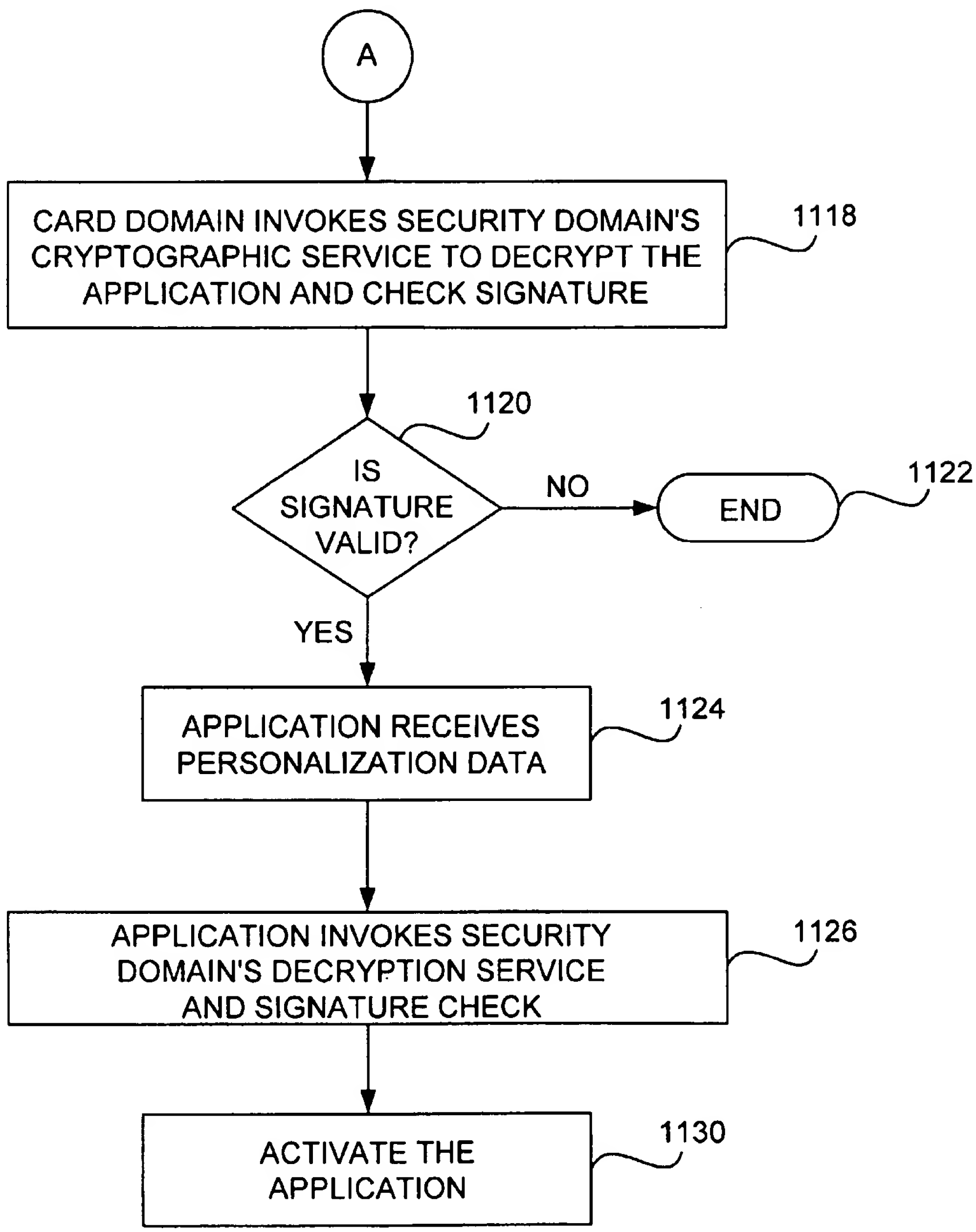

A
CARD DOMAIN INVOKES SECURITY DOMAIN'S CRYPTOGRAPHIC SERVICE TO DECRYPT THE APPLICATION AND CHECK SIGNATURE
1118
1120
IS SIGNATURE VALID?
NO
END
1122
YES
APPLICATION RECEIVES PERSONALIZATION DATA
1124
APPLICATION INVOKES SECURITY DOMAIN'S DECRYPTION SERVICE AND SIGNATURE CHECK
1126
ACTIVATE THE APPLICATION
1130

FIG. 11B

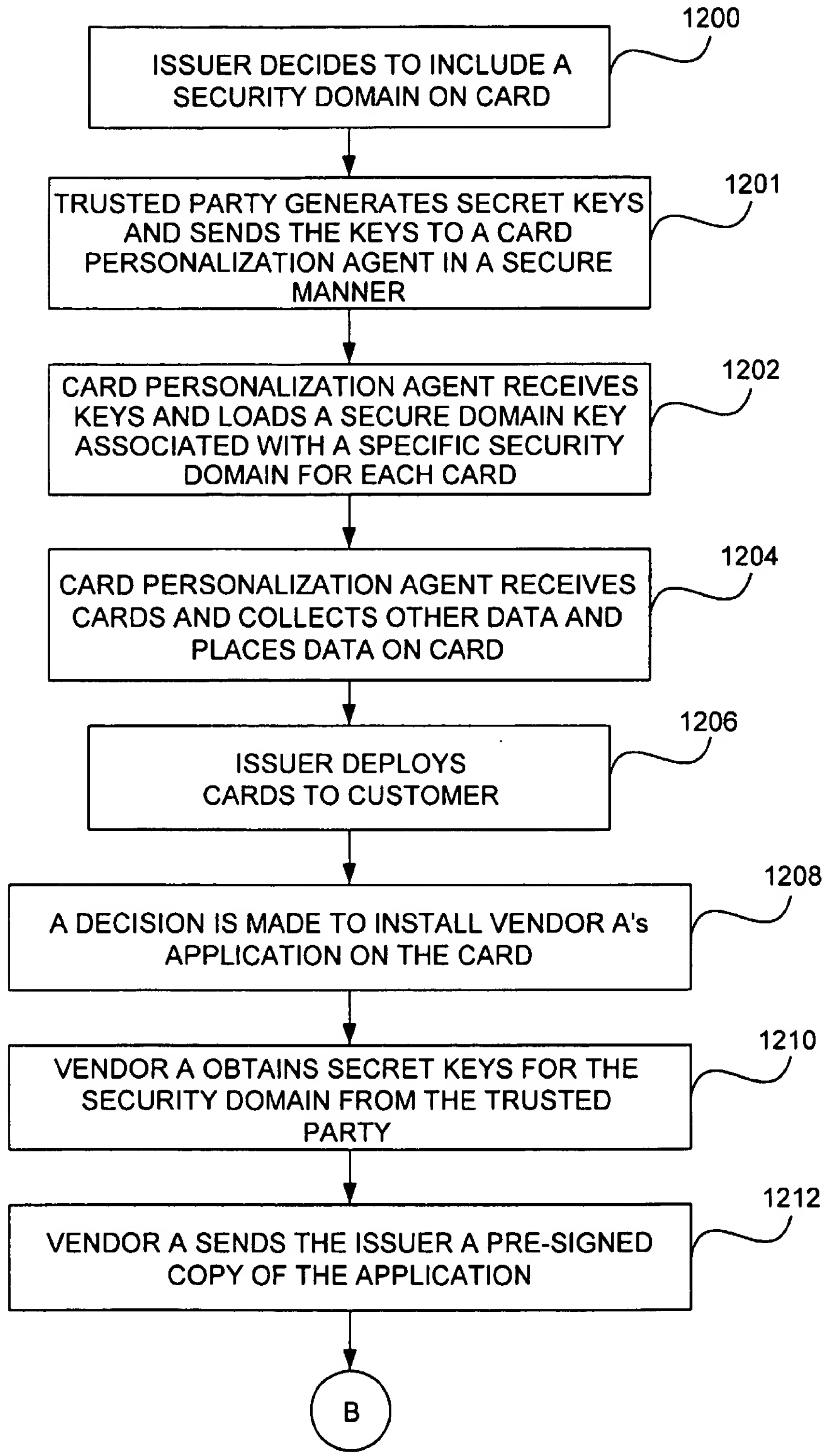

1200
ISSUER DECIDES TO INCLUDE A SECURITY DOMAIN ON CARD
1201
TRUSTED PARTY GENERATES SECRET KEYS AND SENDS THE KEYS TO A CARD PERSONALIZATION AGENT IN A SECURE MANNER
1202
CARD PERSONALIZATION AGENT RECEIVES KEYS AND LOADS A SECURE DOMAIN KEY ASSOCIATED WITH A SPECIFIC SECURITY DOMAIN FOR EACH CARD
1204
CARD PERSONALIZATION AGENT RECEIVES CARDS AND COLLECTS OTHER DATA AND PLACES DATA ON CARD
1206
ISSUER DEPLOYS CARDS TO CUSTOMER
1208
A DECISION IS MADE TO INSTALL VENDOR A's APPLICATION ON THE CARD
1210
VENDOR A OBTAINS SECRET KEYS FOR THE SECURITY DOMAIN FROM THE TRUSTED PARTY
1212
VENDOR A SENDS THE ISSUER A PRE-SIGNED COPY OF THE APPLICATION
B

FIG. 12A

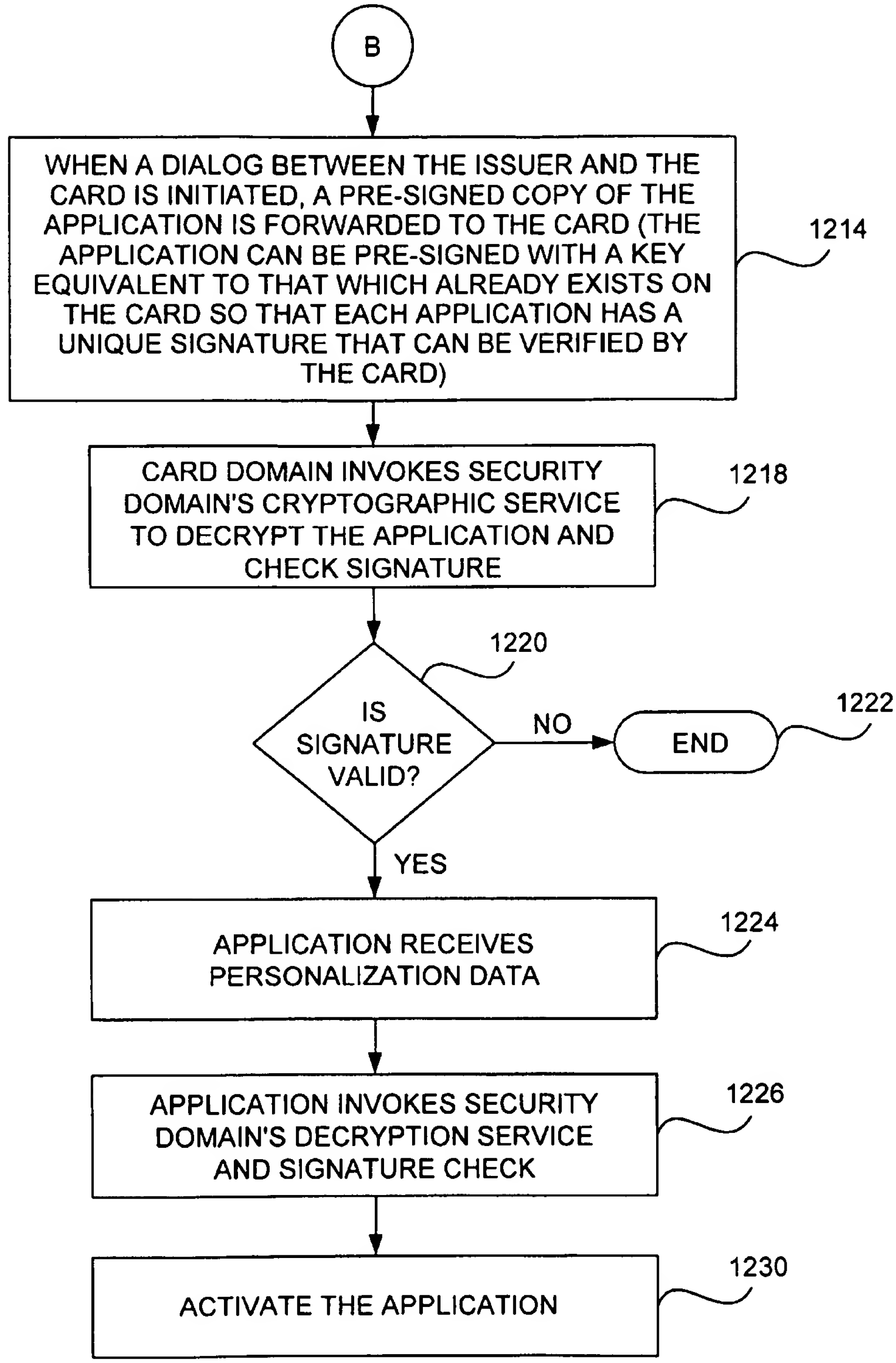

B
WHEN A DIALOG BETWEEN THE ISSUER AND THE CARD IS INITIATED, A PRE-SIGNED COPY OF THE APPLICATION IS FORWARDED TO THE CARD (THE APPLICATION CAN BE PRE-SIGNED WITH A KEY EQUIVALENT TO THAT WHICH ALREADY EXISTS ON THE CARD SO THAT EACH APPLICATION HAS A UNIQUE SIGNATURE THAT CAN BE VERIFIED BY THE CARD)
1214
CARD DOMAIN INVOKES SECURITY DOMAIN'S CRYPTOGRAPHIC SERVICE TO DECRYPT THE APPLICATION AND CHECK SIGNATURE
1218
1220
IS SIGNATURE VALID?
NO
END
1222
YES
APPLICATION RECEIVES PERSONALIZATION DATA
1224
APPLICATION INVOKES SECURITY DOMAIN'S DECRYPTION SERVICE AND SIGNATURE CHECK
1226
ACTIVATE THE APPLICATION
1230

FIG. 12B

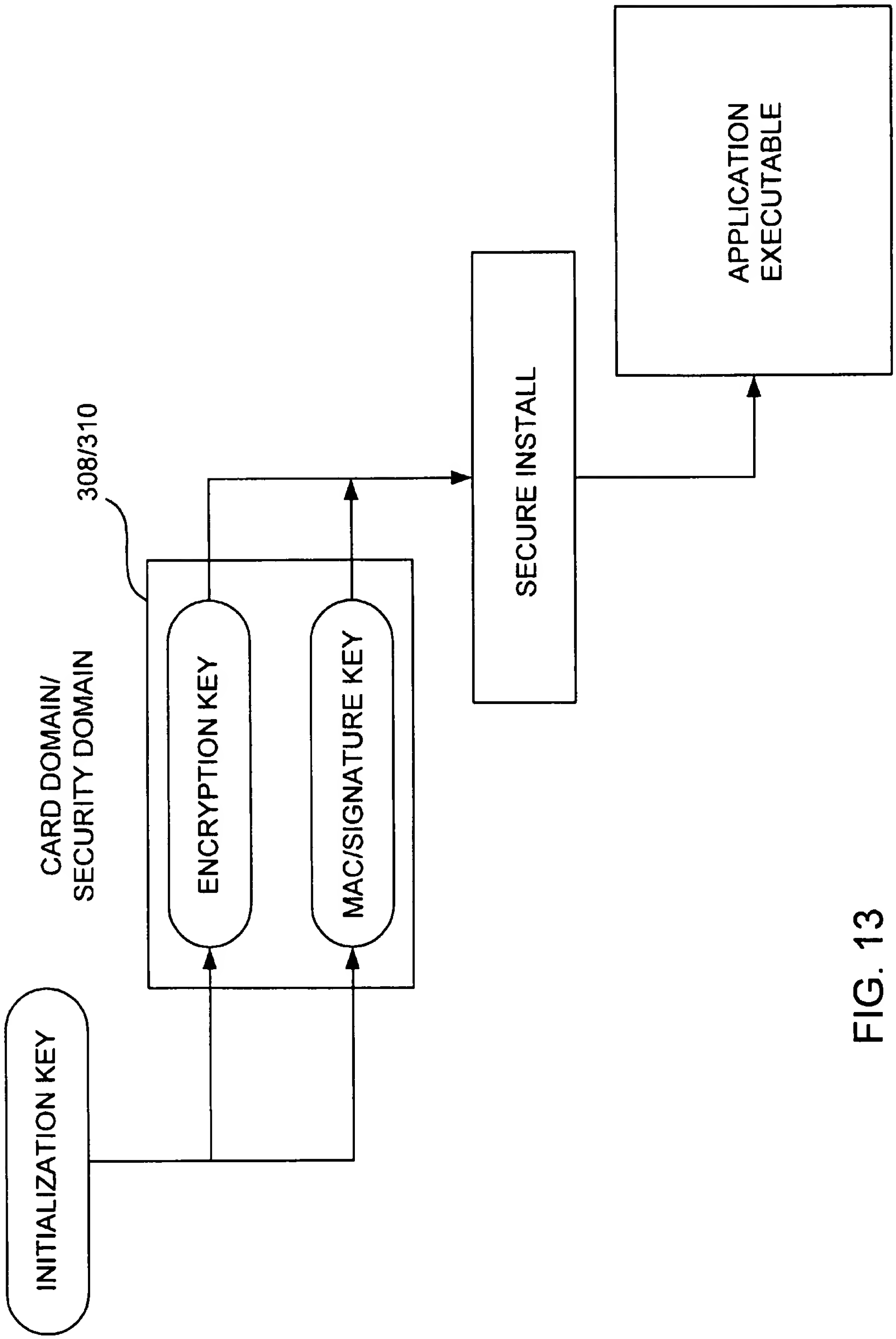

INITIALIZATION KEY
CARD DOMAIN/
SECURITY DOMAIN
308/310
ENCRYPTION KEY
MAC/SIGNATURE KEY
SECURE INSTALL
APPLICATION
EXECUTABLE

FIG. 13

SYSTEM AND METHOD FOR A MULTI-APPLICATION SMART CARD WHICH CAN FACILITATE A POST-ISSUANCE DOWNLOAD OF AN APPLICATION ONTO THE SMART CARD

CROSS REFERENCE TO RELATED APPLICATIONS

This application claims priority to U.S. provisional application Ser. No. 60/061,763 filed Oct. 14, 1997, which is herein incorporated by reference. This application further claims priority to U.S. provisional application Ser. No. 60/041,468 filed Mar. 24, 1997, which is also herein incorporated by reference.

This application is related to U.S. application Ser. No. 09/046,994, filed on Mar. 24, 1998 which is also herein incorporated by reference for all purposes.

FIELD OF THE INVENTION

The present invention relates to smart cards. In particular, the present invention relates to a system and method for providing a multi-application smart card which can facilitate a post-issuance download of an application onto the smart card.

BACKGROUND OF THE INVENTION

A smart card is typically a credit card-sized plastic card that includes a semiconductor chip capable of holding data supporting multiple applications.

Physically, a smart card often resembles a traditional "credit" card having one or more semiconductor devices attached to a module embedded in the card, providing contacts to the outside world. The card can interface with a point-of-sale terminal, an ATM, or a card reader integrated into a telephone, a computer, a vending machine, or any other appliance.

A micro-controller semiconductor device embedded in a "processor" smart card allows the card to undertake a range of computational operations, protected storage, encryption and decision making. Such a micro-controller typically includes a microprocessor, memory, and other functional hardware elements. Various types of cards are described in "The Advanced Card Report: Smart Card Primer", Kenneth R. Ayer and Joseph F. Schuler, The Schuler Consultancy, 1993.

One example of a smart card implemented as a processor card is illustrated in FIG. 1. Of course, a smart card may be implemented in many ways, and need not necessarily include a microprocessor or other features. The smart card may be programmed with various types of functionality, including applications such as stored-value; credit/debit; loyalty programs, etc.

In some embodiments, smart card **5** has an embedded micro-controller **10** that includes a microprocessor **12**, random access memory (RAM) **14**, read-only memory (ROM) **16**, non-volatile memory **18**, a cryptographic module **22**, and a card reader interface **24**. Other features of the micro-controller may be present but are not shown, such as a clock, a random number generator, interrupt control, control logic, a charge pump, power connections, and interface contacts that allow the card to communicate with the outside world.

Microprocessor **12** is any suitable central processing unit for executing commands and controlling the device. RAM **14** serves as storage for calculated results and as stack memory. ROM **16** stores the operating system, fixed data, standard routines, and look up tables. Non-volatile memory **18** (such as EPROM or EEPROM) serves t o store information that must not be lost when the card is disconnected from a power source but that must also be alterable to accommodate data specific to individual cards or any changes possible over the card lifetime. This information might include a card identification number, a personal identification number, authorization levels, cash balances, credit limits, etc. Cryptographic module **22** is an optional hardware module used for performing a variety of cryptographic algorithms. Card reader interface **24** includes the software and hardware necessary for communication with the outside world. A wide variety of interfaces are possible. By way of example, interface **24** may provide a contact interface, a close-coupled interface, a remote-coupled interface, or a variety of other interfaces. With a contact interface, signals from the micro-controller are routed to a number of metal contacts on the outside of the card which come in physical contact with similar contacts of a card reader device.

Various mechanical and electrical characteristics of smart card **5** and aspects of its interaction with a card reading device are defined by the following specifications, all of which are herein incorporated by reference.

Visa Integrated Circuit Card Specification, (Visa International Service Association 1996).

EMV Integrated Circuit Card Specification for Payment Systems, (Visa International Service Association 1996).

EMV Integrated Circuit Card Terminal Specification for Payment Systems, (Visa International Service Association 1996).

EMV Integrated Circuit Card Application Specification for Payment Systems, (Visa International Service Association 1996).

International Standard, Identification Cards—Integrated Circuit(s) Cards with Contacts, Parts 1–6 (International Standards Organization 1987–1995).

Prior to issuance of a smart card to a card user, the smart card is initialized such that some data is placed in the card. Initialization refers to the population of non-volatile memory with data that is common to a large number of cards while also including a minimal amount of card unique terms (e.g. card serial number and personalization keys). For example, during initialization, the smart card may be loaded with at least one application, such as credit or stored cash value, a file structure initialized with default values, and some initial cryptographic keys for transport security. Once a card is initialized, it is typically personalized. During personalization, the smart card is loaded with data which uniquely identifies the card. For example, the personalization data can include a maximum value of the card, a personal identification number (PIN), the currency in which the card is valid, the expiration date of the card, and cryptographic keys for the card.

A limitation of conventional smart cards is that new applications typically can not be added to an issued smart card. Smart cards are traditionally issued with one or more applications predefined and installed during the manufacturing process of the card. As a result, with traditional smart card implementation, once a card has been issued to a card user, the smart card becomes a fixed application card. If a new application is desired, the smart card is typically discarded and a new smart card, which includes the new application, is issued.

It would be desirable to provide a smart card which would allow applications to be loaded after the card is issued.

Further, it is desirable to provide a mechanism to manage the loading of an application as well as general management of the applications on the smart card. Additionally, it is desirable to allow an application provider to keep cryptographic keys confidential from the issuer of the smart card and to securely allow applications from different entities to coexist on a card.

SUMMARY OF THE INVENTION

Embodiments of the present invention teach a system and method which allow card issuers to add applications during the lifetime of the card after the card has already been issued (referred to herein as post issuance loading). Downloading an application after the card has been issued to the card holder will be referred to herein as a "secure install" process.

The system and method according to embodiments of the present invention allow the post issuance loading of an application and/or objects from an application server via a card acceptance device and its supporting system infrastructure delivery mechanism onto a card in a secure and confidential manner.

An embodiment of the present invention provides a system and method for controlling at least one function associated with an issued smart card. In a multiapplication smart card, a privileged application, herein referred to as a card domain, manages multiple functions related to the smart card. Examples of these functions include card initialization, global card data, card life cycle, and secure installation of smart card applications.

A method according to an embodiment of the present invention for providing a first application onto an issued smart card comprises the steps of forwarding the first application to the issued smart card; and loading the first application onto the issued smart card, wherein the loading of the first application is managed by a second application.

In another aspect of the invention, a system according to an embodiment of the present invention for controlling at least one function associated with an issued smart card is disclosed. The system comprises a first application associated with the issued smart card; and a second application associated with the issued smart card, the second application being in communication with the first application, wherein the second application manages at least one function associated with the first application.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a smart card system suitable for implementing the present invention.

FIG. 2 is an example of a block diagram of software layers which can be utilized in a smart card.

FIGS. 3A–3B are block diagrams of examples of software layers according to embodiments of the present invention.

FIG. 4 is a flow diagram of an example of a method according to an embodiment of the present invention for installing an application onto an issued smart card utilizing a card domain.

FIG. 5 is a flow diagram of a method according to an embodiment of the present invention for providing confidential information to an application in a smart card using security domains.

FIG. 6 is a flow diagram of an example of a method according to an embodiment of the present invention for installing an application onto an issued smart card utilizing a card domain.

FIG. 7A is a flow diagram illustrating a sequence of card life states.

FIG. 7B is a flow diagram illustrating a sequence of card life states.

FIG. 8 is an illustration of an example of a card life cycle.

FIG. 9 is a flow diagram of an example of a method according to an embodiment of the present invention for blocking a card utilizing a card domain.

FIG. 10 is a block diagram illustrating interactions between a card domain and a security domain on a smart card according to an embodiment of the present invention.

FIGS. 11A and 11B are flow diagrams of an example of a method according to an embodiment of the present invention for loading an application by using a security domain after the smart card has issued.

FIGS. 12A–12B are flow diagrams of an example of a method according to an alternate embodiment of the present invention for loading an application using a security domain after the smart card has issued.

FIG. 13 is a block diagram illustrating an example of key management and key dependencies for post issuance download of applications onto the smart card.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The following description is presented to enable one of ordinary skill in the art to make and to use the invention and is provided in the context of a patent application and its requirements. Various modifications to the preferred embodiments will be readily apparent to those skilled in the art and the generic principles herein may be applied to other embodiments. Thus, the present invention is not intended to be limited to the embodiment shown but is to be accorded the widest scope consistent with the principles and features described herein.

FIG. 2 is a block diagram of an example of software layers which can be utilized in a smart card. The smart card shown in FIG. 2 includes an operating system 200, a card application programming interface (API) 204, and applications 206A–206B. Operating system 200 can include functionality to control the cards, memory management, input/output (I/O), and cryptographic features. Card API 204 utilizes the instructions from operating system 200 and writes these instructions into blocks which can be reused for common routines in multiple applications. Applications 206A and 206B can run on the smart card via instructions from API 204. These applications can include any application which can run on a smart card, such as stored value, credit, debit, transit, and loyalty.

One embodiment of the present invention is based upon the Java Card standard. In this case applications are referred to as 'Applets' and they are written to link to a Java Card API which is the application programming interface present on smart cards built to the Java Card standard.

Although the conventional software system shown in FIG. 2 allows for multiple applications, it does not solve the problem of how to securely load an application after issuance of the smart card to a user. If an application is to be loaded post issuance, a mechanism is needed to manage the loading of an application as well as the general management of the applications on the smart card. Additionally, an application provider may wish to keep cryptographic keys confidential from the issuer of the smart card. Accordingly, a mechanism is needed to provide for the separation of confidential information between an application provider and an issuer of a smart card. Embodiments of the present invention address such a need.

FIGS. 3A–3B are block diagrams showing software components of a smart card according to embodiments of the present invention. The arrows indicate dependencies between components. FIG. 3A shows an embodiment of a smart card utilizing a card domain, while FIG. 3B shows an embodiment of a smart card utilizing a security domain, as well as a card domain.

The example shown in FIG. 3A includes an operating system **300**, a card API **304**, applications **305A–305C**, a card domain **308**, and open platform (OP) API **306**. The system shown in FIG. 3 allows for a secure and managed post issuance download of an application onto a smart card. A card domain is a card issuier's on-card control mechanism for a smart card according to the present invention.

Open platform API **306** classifies instructions into card domain **308** and security domains **310A–310B** (shown in FIG. 3B). Accordingly, OP API **306** facilitates the formation of instructions into sets which can be identified as being included as part of card domain **308** and security domains **310A–310B**.

Applications **305A–305C** can include any application which can be supported by a smart card. Examples of these applications include credit, debit, stored value, transit, and loyalty. Applications **305A–305C** are shown to include command interfaces, such as APDU interfaces **354A–354C** which facilitate communication with the external environment. APDU stands for "Application Protocol Data Unit" and is a standard communication messaging protocol between a card acceptance device and a smart card. A command is a message sent by the terminal to the smart card that initiates an action and solicits a response from the smart card.

Applications **305A–305C** can run on the smart card via instructions from card API **304**. Card API **304** is implemented using the instructions from the card operating system and writes these instructions into blocks which can be reused for common routines for multiple applications. Those skilled in the art will recognize that a translation layer or interpreter may reside between API **304** and operating system **300**. An interpreter interprets the diverse hardware chip instructions from vendor specific operating system **300** into a form which can be readily utilized by card API **304**.

Card domain **308** can be a "privileged" application which represents the interests of the smart card issuer. As a "privileged" application, card domain **308** may be configured to perform multiple functions to manage various aspects of the smart card. For instance, card domain **308** can perform functions such as installing an application on the smart card, installing security domains **310A–310B** (shown on FIG. 3B), personalization and reading of card global data, managing card life cycle states (including card blocking), performing auditing of a blocked card, maintaining a mapping of card applications **305A–305C** to security domains **310A–310B**, and performing security domain functions for applications **305A–305C** which are not associated with a security domain **310**.

Card domain **308** is shown to include an API **350** and a command interface, such as Application Protocol Data Unit (APDU) interface **352**. APDU interface **352** facilitates interfacing with the external environment in compliance with, e., International Standards Organization (ISO) Standard 7816-4, entitled "Identification Cards—Integrated circuit(s) cards with contacts—Part **4**, Inter-industry commands for interchange," which is herein incorporated by reference.

For example, APDU interface **352** can be used during post issuance installation of an application or during loading of card global data. An application load and install option is performed via a set of appropriate APDU commands received by card domain **308**. API **350** facilitates interfacing with the internal smart card environment. For example, API **350** can be used if card domain **308** is being utilized as a default in place of a security domain **310**, or if an application requires information such as card global data, key derivation data, or information regarding card life cycle. In other words, Card Domain **308** via API **350** also processes, APDUs for functions such as: reading ICC serial number, managing the card life cycle state including providing a card blocking service (the issuer is responsible for determining which applets, if any, can use the card blocking service), performing auditing for the card (when the card is blocked these are the only APDUs that will be handled), maintaining a mapping of security domains to applets, and acting as the security domain for the issuer's applets.

Memory allocations have been performed by the time an application is in an install state. An application is also personalized after loading and installing. A personalized application includes card holder specific data and other required data which allows the application to run. In addition to managing the installation and personalization of the application, card domain **308** can also manage global card information. Global card information includes information that several applications may need to perform their functions, such as card holder name and card unique data utilized in cryptographic key derivations. Card domain **308** can be a repository for the global card information to avoid storing the same data multiple times.

Card domain **308** can also manage card life cycle states including card blocking. The smart card will typically move through several states during its life cycle. Card domain **308** keeps track of what state the card is in during its life cycle. Card domain **308** may also manage a block request to block virtually all functions of the card. Further details of card domain **308** management of a block request will be discussed in conjunction with FIG. **6**. Card domain **308** may also keep track of the state of an application during an application's life cycle. This kind of information regarding an application can be utilized during an auditing of a card. Auditing can be performed at any time during a card's lifetime. For instance, auditing may be performed after a card has been blocked or prior to installing a new application to validate the card contents. Although virtually all card functions are no longer functioning when a card is blocked, an issuer may be able to query card domain **308** for information regarding a state of an application or the life cycle state of the card. In this manner, the issuer of a card may still access a profile of the blocked card and its applications.

FIG. 3B shows an embodiment of the present invention utilizing a security domain **310**, as well as card domain **308'**. The example shown in FIG. 3B includes an operating system **300'**, a card API **304'**, applications **305A'–305C'**, security domains **310A–310B**, a card domain **308'**, and open platform (OP) API **306'**. ~~The system shown in FIG. 3B also allows for a secure and managed post issuance download of an application onto a smart card.~~

Card domain **308'** can work in conjunction with a security domain **310**. Security domain **310** is a logical construct that can be implemented as an application to provide security related functions to card domain **308'** and to applications associated with security domain **310**. Security domains **310A–310B** can assist in secure post issuance loading of an application onto the smart card. Security domains **310A–310B** provide for a mechanism which keeps the

application provider's confidential information, such as cryptographic keys, from being disclosed to the issuer of the smart card.

There may be multiple security domains **310** on a smart card, each represented by a unique cryptographic relationship. A security domain **310** is responsible for the management and sharing of cryptographic keys and the associated cryptographic methods which make up the security domain's cryptographic relationship. An application which is loaded to the smart card post issuance can be associated with a security domain, preferably with only one security domain. However, multiple applications may be associated with the same security domain **310**. Applications installed on a smart card during the pre-issuance phase may optionally be associated with a security domain **310** on the smart card for purposes of loading confidential personalization data to those applications using security domain **310** keys. A security domain is the logical representation of an applet provider's encryption and signature keys on a smart card according to the present invention.

The software for security domain **310** may be installed by the card manufacturer at the time of card manufacturing (e.g., when the ROM is masked), or may be added during initialization or personalization stages. Security domains **310** can be implemented as selectable applications which are isolated from one another and the rest of the system. If security domain **310** is implemented in a Java card as an application, standard Java card security can be relied upon to ensure isolation of security domain **310**. In addition, or alternatively, other security mechanisms such as hardware security can be utilized through OP API **306** implementation. OP API **306** may utilize special security features to enforce isolation of security domain **310**. An example of such a security feature is the utilization of chip hardware security routines which may be employed by OP API **306**.

Each security domain **310A–310B** provides a command interface, such as an Application Protocol Data Unit (APDU) interface **320A–320B**, for communication off card, and on card APIs **322A–322B**.

The APDU interface **320A** or **320B** consists of personalization commands and is intended to allow the initial loading of security domain keys and to support key rotation if desired during the life of the security domain. APIs **322A–322B** may include a signature verification method and decryption method which are shared with card domain **308'** for post issuance loading of applications. Additionally, applications may utilize API interfaces **322A–322B** for decrypting application confidential data. Note that card domain **308'** may always function as a security domain and does so as the default.

Security domain **310** manages signing and decrypting keys and provides cryptographic services using those keys. Security domain **310** processes APDU's for numerous functions. These functions can include key management functions, e.g., functions to load or update keys. During a secure installation of an application, security domain **310** can provide services to card domain **308'** to decrypt an application install file and check the signature of an application file. For an application associated with a security domain **310**, that application's security domain **310** provides decrypt and signature functions, such as MACing on an update key APDU command during the personalization phase of a newly installed application. Thereafter, the application can use the updated key to decrypt and check signatures on subsequent key updates. An install file is a specific data construct format for downloading data onto smart cards on a post issuance basis. MAC stands for "Message Authentication Code" and is a symmetric cryptographic transformation of data that protects the sender and recipient of the data against forgery by third parties.

The smart card issuer may decide whether security domain **310** utilizes a static key or a session key for transactions. A static key is a cryptographic key which exists prior to processing APDUs and which exists during and after the processing of APDUs. A session key is a cryptographic key which can be generated for a particular transaction and is typically no longer used for APDU processing after the transaction. If a session key is utilized, security domain **310** preferably derives its own session key for processing APDUs.

The APDU interface implemented by security domains consists of the initialization and personalization command set. These commands are intended to load security domains, to allow the initial loading of keys and to support key rotation. The general sequence of APDUs to establish one security domain is: 1. SELECT (Card Domain); 2. Perform authorization; 3. INSTALL (SecurityDomain); 4. Multiple LOAD (applet—executable); 5. INSTALL (install—and—register); 6. SELECT (SecurityDomainID); 7. PUT—KEY (SD encryption key encrypted with Card Domain initialization key); 8. Optional PUT—KEY (SD signature key encrypted with SD encryption key).

JAVA EMBODIMENT

One embodiment of the present invention makes use of the JAVA Card standard. The foundation of the card architecture is the JAVA Card virtual machine. The virtual machine executes byte code and manages class and objects. It also enforces separation between applications and enables secure data sharing. Above the virtual machine is the JAVA Card 2.0 API which provides a high level framework for writing applets for JAVA Card based platforms. These classes provide a high level language interface to the underlying functionality required by card applets which include the APDU buffer, commit/role back functionality, data sharing, and life cycle state. The classes are implemented in a mix of JAVA and native methods (i.e., assembly language) enabling applets to be implemented in pure JAVA without using native methods. This distinction enables easier verification of applet behavior and enforcement of security. It also means that the primitive functionality that applets require is contained in this API. In this embodiment, open platform API **306** provides base classes for the development of applets, security domains, and the card domain. OP API **306** describes additional packages for supporting the OP environment on a JAVA card. These packages provide the foundation for implementing the Card Domain applet and Security Domain applets.

Also included in this embodiment is a Card Executive which is responsible for dispatching APDUs to the current applet and selection of the current applets. The Card Executive selects an applet to become the current applet when a SELECT APDU is received. The Card Executive dispatches a non SELECT APDU by calling the current applets process method.

When receiving a SELECT APDU a Card Executive acts as follows. If the application identifier in the SELECT APDU belongs to a selectable applet (i.e., an applet in the personalized STATE) the Card Executive makes that applet the current applet which will receive subsequent APDUs. The Card Executive calls the applets select method. Notice that this method enables the current applet to handle

SELECT APDUs as long as the application identifier in the APDU is not for another selectable applet.

The Card Domain is a special system applet representing the card issuer. It provides platform wide (or "card global") services. The Card Domain processes APDUs for the following functions: installing applets on the card; personalizing and reading card global data, such as card serial number; managing the card life cycle state including providing a card blocking service (the issuer is responsible for determining which applets, if any, can user the card blocking service); performing auditing for the card, when the card is blocked these are the only APDUs that will be handled; maintaining a mapping of security domains to applets; and acts as the security domain for the issuer's applets.

A Security Domain is a special applet owned by an applet provider to control the installing of the provider's applets. It manages signing and decrypting keys and provides cryptographic services using those keys. A Security Domain processes APDUs for the following functions: key set load; key update; and key set switch. During secure install of an applet, the Security Domain provides services to the card domain to decrypt an applet install file and check the signature on an applet file. For an applet associated with a Security Domain, the Security Domain provides decrypt and MAC functions on the Update Key APDU during the personalization phase of the newly installed applet. This is done exactly once: thereafter the applet can use the updated key to decrypt and check the MAC on the subsequent Key Updates. A Security Domain derives it's own session key for processing APDUs.

The virtual machine and Card Executive must be functional before any applet can be selected. As part of card initialization the Card Domain applet is installled and registered. It can then be selected and personalized (e.g. with card global data and it's own keys). After that it can be selected and used to install other applets. Card initialization includes the following steps: the Card Domain is installed and registered during the bootstrapping process of the first time boot of the Card Executive; the Card Domain is selected and the Security Domains from ROM are installed using the INIT key of the Card Domain; select each Security Domain and update their keys using INIT key of the Card Domain; select the Card Domain and load any card personalization data; and select the Card Domain and install applets, some or all of which may be in ROM.

The card security architecture allows for multiple applet owners to exist on card via the use of Security Domains. A Security Domain is a logical construct that represents the aggregation of applets that are owned by a common entity or owner. That common owner is represented on the card by the existence of a cryptographic relationship that has been securely loaded into the card and is supported by the Card Domain. A Security Domain establishes and maintains a chain of trust to a card, via this cryptographic relationship, for an applet owner. There may be multiple, Security Domains on a card with each represented by a unique cryptographic relationship.

The purpose of a Security Domain is to enable the post issuance loading of applets and confidential applet data using the cryptographic keys associated with that Security Domain. Note that the use of secure messaging during the post issuance load process is independent of the cryptographic functionality of the Security Domains. This cryptographic relationship includes the association of a cryptographic key with its owner and identifies specific applets or a range of applets that are controlled by the owner. The existence of this cryptographic relationship allows more than one applet owner to be represented on the card. Each owner, based upon their control of their cryptographic relationship is thus responsible for the security and integrity o f all applets within their Security Domain.

POST-ISSUANCE LOADING

FIG. **4** is a flow diagram of a method accordingly to an embodiment of the present invention for providing an application to a smart card. The example illustrated in FIG. **4** also applies to installing a security domain **310** onto a smart card. Note that all of the flow diagrams in this application are merely examples. Accordingly, the illustrated steps of this and any other flow diagram , can occur in various orders and in varying manners in order to accomplish virtually the same goal.

A smart card is issued (step **400**), and an application is forwarded to the issued smart card (step **402**). The forwarding of the application can occur through any electronic media which can interface with a smart card and connect to an appropriate network. For example, devices such as an automatic teller machine (ATM), a display phone, or a home computer, can be used to forward an application to the issued smart card. The forwarded application is then loaded onto the smart card, and the loading of the application is managed by card domain **308** (step **404**).

FIG. **5** is another flow diagram of a method according to an embodiment of the present invention for providing an application onto an issued smart card. A smart card is created and provided with a first application, the first application including a cryptographic service (step **1002**). A second application is loaded onto the smart card (step **1004**). Thereafter, the second application is installed, and the cryptographic service of the first application is utilized to install the second application (step **1006**).

FIG. **6** is another flow diagram of an example of a method according to an embodiment of the present invention for providing an application onto an issued smart card. This method for providing an application also applies to providing a security domain **310** onto the smart card. In the example shown in FIG. **6**, a card issuer deploys smart cards to customers (step **500**). A decision is made to install a vendor's application onto the issued smart card (step **502**). When a dialogue between the issuer and the smart card is initiated, a pre-signed copy of the application is forwarded to the smart card (step **504**). As previously stated, the dialogue between the issuer and the smart card can occur via any electronic device which can interface with a smart card and connect to an appropriate network. The application can be pre-signed with a key equivalent to that which already exists on the card so that each application has a unique signature that can be verified by the card.

Card domain **308** can then take the steps to load the application. Card domain **308** decrypts the forwarded application and checks the signature of the application (step **508**). Card domain **308** can decrypt the application with the issuer's secret key. An appropriate cryptography method, such as Data Encryption Standard (DES) or 3DES, can be utilized to decrypt at least a portion of the application. Those skilled in the art will recognize that a number of cryptographic techniques may be used to implement embodiments of the present invention. For the purpose of illustration, symmetric key techniques are addressed herein, although asymmetric techniques are also contemplated. A good general cryptography reference is Schneier, Applied Cryptography, 2d Ed. (John Wiley, 1996), the contents of which are incorporated herein by reference.

It is then determined whether the signature on the application is valid (step **510**). If the signature associated with the application is not valid, then the application is not loaded onto the card and the process ends (step **520**). If, however, the signature associated with the application is valid the application is then installed and available for personalization. During personalization the application receives personalization data (step **512**). Personalization data includes data which is unique to the smart card user. For instance, in a airline loyalty application, personalization data can include the smart card user's seating preference, meal preference, and eligibility for various possible perquisites. This personalization data can also be signed and encrypted.

During personalization, cardholder and card-specific data are placed on the card. This data is unique to the card and unique to the individual applets. Because only applets can be personalized, the traditional term "card personalized" is no longer appropriate for Open Platform cards. Applet personalization is performed under the complete control of each applet. It is the responsibility of each applet to provide proper auditing information regarding applet life cycle state to the Card Domain including informing the Card Domain once personalization is complete.

Every applet is personalized separately. Personalization data is always owned by a specific applet. Personalization of an applet can begin once the applet has been registered. Each applet receives APDU-level commands for personalization. These commands can be Update commands which the applet uses to populate its field structures with the appropriate data.

Each applet is responsible for determining when applet personalization is complete so that the applet state can be set to "personalized." Applets implement the APDU commands Set Status to finalize personalization and share the information appropriately with the Card Domain to allow auditing.

The following list illustrates the sequence of APDU commands that is used when a personalization key is involved. Personalization of an applet is started by selecting it with the Select command. Initialize Update is executed to allow the applet to respond with the relevant derivation data that has been loaded during initialization. The session key is calculated based on the current personalization key. External Authenticate is then issued to establish the appropriate level of trust. The personalization key that is initially used by an applet may have been put on the card during initialization. This personalization key may then be replaced with the "real" personalization key via the Put Key command. The new key data is encrypted using the current session key.

Initiate is again issued resulting in a new session key calculated based on the unique personalization key just loaded previously. A new session key has been established and it is used for all following personalization commands. The final step of personalizing is the APDU command Set Status.

The application then invokes card domain's **308** decryption service for the received personalization data (step **513**). Card domain **308** can then performs a signature check for the received personalization data (step **514**). Methods of decrypting personalization data and performing signature checks are well known in the art. Finally, the application can then be activated (step **518**).

A new application which as been downloaded onto a smart card post-issuance can be stored in a variety of ways. One example is to store the application into a file. Another example is to maintain a pointer to the application object.

CARD AND APPLET LIFE CYCLE

The life cycle of a card and/or applet is defined as the time the integrated circuit and/or applet is first defined, to the time that the card is disposed of, or the applet is deleted by the cardholder or card issuer. Throughout this life cycle, the Open Platform card and its applets will transition through a number of pre-defined states. Each state that is reached will enable the card and/or applet to perform a fixed number of tasks. The Open Platform card provides a multi-applet architecture for chip cards and as such must take into account the life cycle of the card as a whole, and the life cycle of each applet contained within the card. The Card Domain is the entity on a card that is responsible for the card life cycle state management.

FIG. 7A is a flow diagram illustrating an example of card life cycle. The sequence is preferably considered irreversible. The first card state is when the smart card is Masked (**700**). During the Masked state (**700**), the smart card obtains, its operating system, card identification, and preferably at least one application. The Masked state (**700**) is achieved as soon as all of the necessary components for card initialization are made available. An example of when necessary components are made available is when card domain **308** and OP API **306** are enabled, as well as the Java card environment being enabled, such as a Java card virtual machine and a Java card API. During masking, the operating system, virtual machine, class libraries and optionally, applets are written into the non-mutable memory of a chip.

After the Masked state, the next state is the Initialized (**702**) state. The Initialized state is achieved once all card activity requiring an initialization key is complete. As part of card initialization, if not already available, the card domain **308** application must be installed and registered. In addition, one or more security domains may also be installed and registered. These installed domains must then be selected and personalized. An initialization key is a secret key which is typically used by a smart card manufacturer during loading of data onto the smart card prior to issuance.

The next state is Load Secured (**704**). The Load Secured state is achieved after a secure install (post-issuance download) mechanism for loading of applications through the remainder of the card lifetime has been established.

The final card state is when the card is either expired or blocked (**706**). The blocked state is achieved as soon as an authorized smart card application has received a command to block the card.

The card life cycle is preferably an irreversible sequence of states with increasing security. Initialized and all subsequent card life cycle states and their transitions are preferably under the control of card domain **308**. Card domain **308** executes and responds to commands that result in a transition in the card life cycle from one state to the next. These commands are preferably Application Protocol Data Unit (APDU) commands. Card domain **308** is also responsible for the installation of applications on the card, but preferably has no control over the applications' life cycle states. Each application is preferably responsible for its own application life cycle state management but it preferably allows card domain **308** to have access to its life cycle states for auditing purposes.

The card life cycle is designed in such a way to increase the level of security enforced by the card at each successive state. As stated above, the cycle is also established as a process which can only ratchet forward to ensure that once the card begins a life cycle state with associated security policies, the only option is to cycle forward to the next state in the life cycle with a higher level of security. The card domain as the system security manager of the card maintains the current life, cycle state, enforces the associated security policies, and controls the state transitions in the card life cycle.

FIG. **7B** is a flow diagram illustrating an example of an application life cycle. The application is initially unavailable (**750**). This state is achieved as soon as an applet is deleted and/or blocked from access. The next state is a loaded state (**752**). The application reaches the loaded state once the application has been physically loaded onto the smart card. The application is then installed (**754**), and registered (**756**). The installed state is achieved as soon as an applet has allocated all of the necessary space and data structures for operation. The registered state is achieved as soon as an applet has been included in the registry of applets. Once the application is registered, it can be deleted at any time thereafter. The next state is the personalized state, wherein personalized information is included in the application (**758**). The personalized state is achieved as soon as an applet has been updated with card holder specific data. Finally, the application may expire or be blocked (**760**). The blocked state is achieved as soon as the applet makes a distinctive call to block itself from further access. The card domain takes responsibility for loading, installing, registering and deleting applets. The applet itself is responsible for handling of the applet life cycle states "personalized" and "blocked." An applet is not required to transition through all states during its life. For example, an applet may be present on the card but remain dormant throughout the card lifetime because its install method is never called. The Card Domain takes responsibility for and controls the loading, installation and registration process of an applet until the applet is selectable. The Card Domain keeps track of the process and can report on its status. The Card Domain does not control the applet life cycle states "personalized" and "blocked", because these applet life cycle states are under the complete authority of the applet only. However, the Card Domain does have access to applet life cycle data for auditing purposes.

The life cycle of the card and its applets are determined by the contents of the card memory. The smart card contains three types of memory: persistent, non-mutable memory; persistent mutable memory; and non-persistent mutable memory. Although there are several different types of memory technologies currently available for each of the memory types, ROM, EEPROM and RAM are the most widely used for these respective types. This section uses these abbreviations when referring to the memory types, but does not imply or mandate the use of a certain physical memory technology.

The contents of the ROM are defined first. The ROM contains the card operating system and may also contain applet code. More applets may be added during the life of the card, either during the initialization and the personalization phase, or during the post issuance phase. Loading an applet after the card has been issued is referred to as "Secure Install". The contents of the EEPROM are determined by a memory loading process that occurs after chip fabrication. Therefore, the contents of the EEPROM memory, card life cycle and applet life cycle are closely related because of the dynamics involved. The contents of EEPROM memory are modified during initialization, personalization and Secure Install. Although EEPROM memory contents and card state are closely related, there is nothing in the architecture that ensures that the two are synchronized.

FIG. **8** is an illustration of an example of multi-application card life time line. The illustration does not include any off-card activities that occur prior to masking, initialization, personalization, Secure Install, nor activities that may be related to card management procedures. The illustration shows a card that contains six applications (A through F) over its lifetime.

This time line starts with a Masked ROM stage **800** and ends with a card blocked/expired stage **802**. At Masked ROM stage **800**, applications A, B, C and D are shown to be installed. This example shows applications A and B being installed at a masking stage of the card, applications C and D being installed at initialization stage, and applications E and F being installed post issuance.

In this example, application A can be installed in ROM and used during the complete life of the card from Masked ROM stage **800** to card blocked/expired stage **802**. Application B is also in ROM and utilized during a first portion of the life of the smart card. The life of application B ends at stage **804**. Application C is located in non-volatile memory, such as EEPROM, which is loaded during initialization. Application C is shown to expire at stage **806**. Application D is also located in EEPROM and is used for the complete life of the card until card blocked/expired stage **802**. Application E is installed at stage **808**, sometime after issuance of the smart card. Application E is located in EEPROM and used until the end of the card life at card blocked/expired stage **802**. Application F is also installed post issuance at stage **810**, and expires sometime before the end of the card life at stage **812**.

FIG. **9** is a flow diagram of a method according to an embodiment of the present invention for blocking a card. A card be can be blocked if a breach of security is detected by an application. According to an embodiment of the present invention, a smart card can be blocked while an application is in use. A blocked card will no longer operate so that a suspect user cannot utilize any of the applications on the smart card. Blocking is merely one example of the many functions card domain **308** can perform in managing the other applications on the smart card. Examples of other functions include installing an application on the smart card, installing security domains **310A–310B**, personalization and reading of card global data, managing card life cycle states including card blocking, performing auditing of a blocked card, maintaining a mapping of card applications to security domains, and performing security domain functions for applications which are not associated with a security domain.

In the example shown in FIG. **9**, an application is currently in use (step **600**). The application detects a problem which triggers a card block request from the application (step **602**). The application then sends a card block request to card domain **308** (step **604**). Card domain **308** determines whether the card block request is valid (step **606**). A card block request can be valid if the request originate from a predetermined application. If the card block request is not valid, the card domain **308** does not block the smart card (step **608**). However, if the card block request is valid, then card domain **308** authorizes the card blocking (step **610**), and card domain **308** blocks the smart card (step **612**) such that the smart card will reject any attempted transactions for any of the applications on the card.

Card blocking may be implemented as blocking of the applet Card Domain. The Card Domain applet must implement the specific behavior of blocking the applet itself (Card Domain). It must also implement the card issuer specific requirements for blocked card behavior.

FIG. **10** is a block diagram illustrating the use of security domain **310** by the card domain **308**. The method and system according to an embodiment of the present invention allows for multiple application providers to be represented on a smart card in a secure and confidential manner. Th is security and confidentiality can be achieved through the use of security domains **310A–310B** shown in FIG. **3**.

A Security Domain in the context of the present invention architecture is a logical construct that is implemented as an applet to provide security related methods to the Card Domain and to applets belonging to the Security Domain. An applet provider interested in loading this application post issuance must be represented and identified by a Security Domain on the target card. There may be multiple Security Domains on a card, each represented by a unique cryptographic relationship. A Security Domain applet is responsible for the management and sharing of the keys and the associated cryptographic methods which make up the domains cryptographic relationship. An applet which is loaded to the card post issuance must be associated with only one Security Domain, however, multiple applets may be associated with the same Security Domain. Applets installed on a card during the pre issuance phase may optionally be associated with a Security Domain on the card for the purpose of loading confidential data to those applets using these Security Domain keys. Security Domain applets may be included in the mask or added in initialization or personalization. There are selectable applets which are protected from one another and the rest of the system via the standard applet security. Each Security Domain applet provides an APDU interface for communication off-card and an object sharing interface.

The APDU interface consists of personalization commands and is intended to allow the initial loading of Security Domain keys and to support key rotation if desired during the life of the Security Domain. The object interface is comprised of the signature verification method and the decryption methods which are shared with the Card Domain for post issuance loading of applets and with applets for decrypting application confidential data. Note that the card domain with its keys may always function as a Security Domain as well and does so as the default.

FIG. **10** illustrates an example of a smart card which contains two security domains **310A–310B**. In this example, it is assumed that a masked application **305A** from the smart card is associated with a security domain, such as security domain **310A**, and an additional application **305B** will be added post issuance and be associated with a second security domain, such as security domain **310B**. The arrows indicate key relationships between the various smart card entities as will now be described. Masked application **305A** uses key services from security domain **310A** for decrypting confidential data and optionally for full personalization. Card domain **308** uses key services from security domain **3103B** for decrypting and checking the signature of an application loaded post issuance, such as post issuance loaded application **305B**. Post issuance loaded application **305B** uses key services from security domain **310B** for decrypting confidential data and optionally for full personalization.

FIGS. **11A** and **11B** are further flow diagrams of an example for a method according to an embodiment of the present invention for providing an application onto an issued smart card. The card issuer decides to include a security domain **310** onto a smart card (step **1100**). The issuer assigns security domain **310** to vendor A (step **1102**). Vendor A, or an application developer on behalf of vendor A, generates cryptographic keys such as those used in symmetric or asymmetric cryptography operations (step **1104**). Examples of these cryptography operations include encryption, decryption, MACing, hashing, and digital signatures. Examples of cryptographic methods which utilize such keys and are suitable for implementation for the embodiment of the method and system of the present invention include Data Encryption Standard (DES) and 3DES. The card personalization agent receives the keys and loads security domain keys associated with a specific security domain **310** for each smart card (**1106**). The card personalization agent receives smart cards and collects other data, OS, code, and application and card holder specific data, and places the data on the smart card (step **1108**).

The card issuer then deploys the smart card to customers (step **1110**). A decision is then made to install vendor A's application on the smart card (step **1112**). When a dialogue between the smart card issuer and the smart card is initiated, a signed copy of the application is forwarded to the smart card (step **1114**). The application can be signed with a key equivalent to that which already exists on the smart card so that each application has a unique signature that can be verified by the smart card.

The smart card's card domain **308** then takes steps to load the application. Card domain **308** invokes an associated security domain's cryptographic service to decrypt the application and check the signature (step **1118**). It is then determined if the signature is valid (step **1120**). If the signature is not valid, the process ends (step **1122**). If, however, the signature is found to be valid, then the application receives personalization data which can be signed and optionally encrypted (step **1124**). The loaded application then invokes its associated security domain's decryption service and signature check for the received personalization data (step **1126**). Secret keys required to run or operate the application on the smart card are used to activate the application by authentication (step **1130**).

FIGS. **12A** and **12B** are flow diagrams of a method according to another embodiment of the present invention for providing confidential information to an application using a security domain **310**. The issuer decides to include a security domain **310** on a smart card (step **1200**). A trusted party generates secret cryptographic keys and sends the keys to a card personalization agent in a secure manner (step **1201**). A trusted party is typically a third party who performs the function of certifying the source of information, such as a signature. A card personalization agent (which may be the same as the trusted party) receives the key and loads a unique secure domain key associated with a specific security domain **310** for each smart card (step **1202**).

The card personalization agent receives the smart card and collects other data, such as OS, code, and application and card holder specific data, and places the data on the smart card (step **1204**). The issuer then deploys the smart card to its customers (step **1206**). A decision is made to install vendor A's application on the issued smart card (step **1208**). Vendor A obtains secret keys for security domain **310** from the trusted party (step **1210**). Vendor A then sends the smart card issuer a signed copy of Vendor A's application (step **1212**).

When a dialogue between the smart card issuer and the smart card is initiated, a signed copy of the application is forwarded to the smart card (step **1214**). The application can be signed with a key equivalent to that which already exists on the smart card so that each application has a unique signature that can be verified by the smart card. Card domain **308** invokes security domain's cryptographic service to decrypt the associated application and check its signature (step **1218**). It is then determined whether the signature is valid (step **1220**). If the signature is not valid, then the process ends (step **1222**).

If, however, the signature is valid, then the application receives personalization data, which can be signed and optionally encrypted (step **1224**). The loaded application

then invokes security domain's decryption service and signature check for the received personalization data (step **1226**). The cryptographic secret data required to run or operate the application on the card are used to activate the application (step **1230**).

SECURE INSTALL PROCESS EMBODIMENT

The primary object of the Secure Install Process is to load an applet and/or other objects from an applet server via a card acceptance device and its supporting system infrastructure delivery mechanism, on to a card post issuance in a secure and confidential manner. This process begins once the card issuer has converted the applet into the appropriate install format for download.

The applet provider can choose to sign and encrypt the applet Install File. The following description assumes both encryption and signing to be required. First the Install File must be padded to an eight byte boundary. After padding has been done, the Install File is encrypted using the applet provided encryption key. This key is available in the corresponding Card Security Domain and will be used during the Secure Install process to decrypt the loaded applet. Additionally, the applet provider may choose to sign the applet. The applet is signed by calculating the signature over the complete Install File. If the Install File has been encrypted, the signature is calculated using the encrypted file. If the Install File has been not been encrypted, padding is to be performed before calculating the signature. The signature is calculated using the applet provider Security Domain signature key. The Card Domain Secure Install process uses the Security Domain key services to verify the signature of an applet and to decrypt it. Note that neither encryption nor signing are mandatory steps in the process, but are policy decisions of the applet provider and the card issuer. During the Secure Install Process, the signature service of the corresponding Security Domain is used by the Card Domain to verify the signature of the encrypted Install File. Once verified, the Card Domain may then decrypt the Install File using the decryption service of the Security Domain applet. In order to determine the correct Security Domain, the Card Domain receives the application identifier of the applet to be loaded and its associated Security Domain at the beginning of the download process. The Card Domain performs a check to validate that the application identifier is authorized for the proposed Security Domain on the card.

The card issuer by virtue of the Card Domain provides the key services for the Secure Install APDU level transport layer. Since the Secure Install process relies upon the Card Domain key services to provide security and integrity at the transport level, it is the card issuer who is ultimately responsible for the transport layer. A card issuer may be decide upon the use of APDU encryption during Secure Install; however use of the secure messaging during Secure Install is mandatory.

The card issuer uses the Card Domain to provide integrity and confidentiality during Secure Install by means of DES based cryptographic methods. At the APDU level secure messaging is used to verify that all data of an APDU transmission has been received correctly and unmodified, and to insure the confidentiality of data. Secure messaging can be performed as described in the specification *Visa Integrated Circuit Card,Specification Version* 1.3.

The Card Domain Secure Install process offers full error recovery that is independent of the processing stage reached during a Secure Install section. Either the Secure Install succeeds completely or is not performed at all. In order to perform error recovery successfully, the Secure Install process requires the following functional support. If Secure Install is not completed successfully, the card operating system shall take responsibility to reestablish the original memory contents. The secure install protocol does not provide a mechanism for sporadic data transmission errors, e.g. single APDU repeat is not performed at the install protocol level.

A number of reasons could cause a failure in the Secure Install download process. These include the situations when: a single APDU MAC is not verified; an applet signature can not be verified; a memory allocation problem occurs; or a memory write problem occurs (physical memory error). Failure during Secure Install causes a complete abort of the process; therefore all downloaded information is completely erased from the card's memory.

As indicated previously, Secure Install is implemented within the Card Domain; however, prior to the actual Secure Install process, a number of steps prepare, an applet before download. This section will provide an overview of these steps. The following is a description of the off card tasks that are assumed to be performed within the applet development tool kit. Data entities that must be stored on the card are created such as: a process method; an install method; and applet data (file system and tag data). Steps required to create the install file image are the responsibility of the appropriate set of tools. The result of the download preparation process s the Install File.

The Install File will contain sufficient data about its own contents to allow the card operating system and/or Secure Install to successfully install the file contents. The Install File contents will include: the applet name (applet identifier); the applet's memory requirements; any applet cross-linking (fix up) information; and the applet executable code (process method and install method). All data objects (e.g. file system) to be used by an applet must be create d by the applets install method. The file system creation must therefore be coded the applet provider. Populating the file system with applet specific data and with card specific data is the responsibilty of the applet personalization process. For reasons of confidentiality the applet provider may choose to encrypt the Install File. For reasons of integrity the applet provider may also choose to sign the Install File. The signature is implemented as a symmetric key based signature and is verified using the applet provider's Security Domain signature service. This signature is calculated over the complete Install File.

The card acceptance device initiates the Secure Install process by first selecting the card's Card Domain using the APDU command Select. After selecting the Card Domain, authentication and the initialization of cryptographic functions must take place. Authentication means that the card responds with a cryptogram that allows the loader to verify the card identity. Initializing the cryptographic functions means that the smart card as well as the loader must calculate session keys. The APDU command used is the Initialized Update command. When this command is received, the Card Domain shall calculate the session keys for the APDU MAC verification and for the APDU data decryption. The Card Domain shall create a response consisting of a cryptogram for authentication purposes and key diversification data. Following this response, the Card Domain Secure Install process is now ready to receive the Install File on the card.

Successful completion of the previous command is a prerequisite for the Install File transmission. Transmitting

(loading) the Install File is initiated by the APDU Install command. After successful completion of this command, the Install File Loading can be performed using the APDU Load Install File command.

For signature checking and Install File decryption the Card Domain will use the key services supplied by the corresponding Security Domain. Signature checking and Install File decryption is initiated using the Install command. After a successful verification of the applet signature and decryption of its executable, the Card Domain must now call the applet's install method; allocate memory for its data objects; configure data sharing for the applet; and register the applet with the Card Domain. As soon as the applet install method has completed successful, the applet is integrated into the card environment as follows. The applet application identifier is registered in the list of selectable applets and the card auditing information is updated. Registering an applet makes the applet selectable. Only a registered applet can receive APDU commands for processing. After integrating the applet into the card environment, the applet must be personalized. The data objects (e.g. file system) themselves have been created by the applet's install method. The process is identical to an applet personalization process.

The contents of the applet Install File are not only the applet itself, but also information required by the load process. Therefore the Install File contains: the actual applet class, the mandatory data required to perform the download; and the optional data to aid the download process and to improve its performance.

The applet Install File must contain all information that is required by the card in order to receive the applet, store it in non volatile storage and make it ready to run. This mandatory data includes the following: name or identifier of the applet; identification of hardware and software requirements (version of virtual machine, system class and system framework); link references to libraries and classes in ROM that need to be resolved; link references to libraries/class/functions in non volatile that need to resolved; link references within the applet that need to be resolved; fix ups for data references that need to be resolved; entry points for "process" and "install" methods; and proof of ownership, origin, and completeness and correctness. Optional information includes memory requirements, debug information and any potential terminal related information.

The applet development process begins as soon as a concept of the desired applet is formulated. Once formulated, the concept must go through a number of stages before becoming a full fledged working and installable applet. An applet consists of two primary elements; the applet's process method; and the applet's install method. The applet development process (executable) is considered complete after creating and securely delivering the Install File. This development process may vary by organization but in general consists of the following steps: applet requirements gathering and definition; applet specification is developed from requirements; source code is developed from specification; managing applet source code, testing, approving; translate applet source code into card executable code; conversion to card byte code; card byte code verification; code linking and reference resolution; and signature creation.

The results of the translation process is the Install File. The Install File contains the card byte code that is to be executed by the card. The purpose of this file may be one or more of the following: input to the ROM image (masking); input to tho initialization procedure (pre issuance load); and distribution for post issuance load (Secure Install). The Install File format and its contents are target system independent. The result of an applet development process is a target system independent signed Install File that contains the complete information need for loading an applet onto a card. The applet executable is contained within a single Install File. This Install File can be used for any of the following: the masking process; the initialization process; or the Secure Install process.

CRYPTOGRAPH KEYS FOR POST ISSUANCE LOADING

Key management relies on a hierarchy of keys. Keys higher up in the hierarchy are used to securely load additional keys into the card. This means that the highest key in the hierarchy is an initial key. At the IC level there is a chip password and an initialization key. At the next level below, the Security Domain level, there are Security Domain keys. At the next application level there is an application personalization key and application transaction keys. Finally, down at the session level, there are applications session keys. Note that the Card Domain functions as the default Security Domain on the card, thus the Security Domain level noted above includes the card Domain keys. Security Domain keys may have a hierarchical relationship with the initialization key or with the card domain keys depending on the requirements of the Security Domain owner.

The initialization key is available when the card initialization procedure is authorized. It is used as an encryption key to confidentially load secrets, e.g. other keys, onto the card and is initially owned by the Card Domain. The initialization key is used for the encryption of APDU data during initialization and the decryption of Security Domain and encryption keys. A Security Domain and encryption key is a special dedicated version of an encryption key. Each Security Domain has an encryption key and a signature key. The encryption key is used to decrypt the Install File during post issuance applet loading and to decrypt the applet personalization key.

The Security Domain signature key is a special dedicated version of a MAC key. Each Security Domain has a signature key and in one implementation the signature is represented by a MAC. The MAC key is used to verify the Install File signature during post issuance applet loading. The application personalization key is used during applet personalization and is owned by the applet. The key is used to load card and or applet specific data (which can also be keys) into the applet. An application encryption key is used to encrypt and decrypt data, for example, it can be used to encrypt APDU data. Encryption keys are owned by applets and have applet specific uses. Encryption keys are used to decrypt APDU command data and to encrypt APDU response data. An application MAC key is used to create or to verify a MAC. The keys are owned by applets.

In order to load keys of a specific Security Domain applet, the Security Domain applet has to be selected. The Security Domain encryption key may be encrypted with the Card Domain initialization key or optionally with the Card Domain encryption key if initialization had ended. The Security Domain encryption key is used to encrypt the Security Domain signature key. Security Domain signatures must be loaded encrypted using the encryption key of the corresponding security domain. It is not required to perform this function during initialization, it is only required to be performed before any Secure Install. The above completes

the loading of keys into one or more Security Domains. The Security Domain encryption key service is shared to corresponding applets for decryption of confidential applet data. The key services of the Security Domains are also shared with the Card Domain for Secure Install. In order to load confidential data (including keys), an applet can use decryption services of it's corresponding Security Domain. The Card Domain acts as the default Security Domain.

FIG. 13 is a block diagram illustrating the use of cryptographic keys for post issuance loading of an application onto a smart card. Applications that are not masked and not loaded during card initialization stage or personalization stage need their executables downloaded using a secure installation method, such as the post issuance download described in the previous figures. The applications can be loaded using the card domain cryptographic keys. The applications are then decrypted and can have their signature verified using the key services of the corresponding security domain **310**. Therefore, the desired security domain(s) **310** preferably have encryption and signature keys installed prior to the post issuance download of the corresponding application.

In the example shown in FIG. **13**, only one security domain **310** is shown since security domains **310** for other applications are not relevant to illustrate the downloading of a single application. Note that the result of the secure installation is initially a loaded application, which must then be installed, registered and personalized. After loading, the application is installed, preferably by issuing an install APDU command to card domain **308**. An application can be installed when its install method has executed successfully. Memory allocations have been performed by the time an application is in an install state. A loaded application should also be registered. When an application is registered, it is selectable and it is ready to process and respond to APDU commands. Installation and registration may be performed simultaneously by the same APDU command. An application is also personalized after loading. A personalized application includes cardholder specific data and other required data which allows the application to run.

To allow personalization to be performed, the applet uses services of the security domain to load confidential data. The personalization key can be considered to be the confidential data. Note that all post issuance "PUT—KEY" commands must be MACed and encrypted. Further processing now requires the applet to be selected. After selecting the applet any additional applet keys may be loaded using the personalization key for encryption. Here the applet specific personalization rules may be applied.

In the example shown in FIG. **13**, the cryptographic key and MAC/Signature key are shown to be included in the functions of card domain **308**/security domain **310**. If a security domain is associated with the application being loaded, then the security domain will be invoked. However, if no security domain **310** is associated with the application which is being loaded, then the cryptographic key and the signature key of card domain **308** will be utilized. In contrast to the install commands sent to the smart card during the initialization phase, the post issuance install command is not issued in a secured environment, therefore it is preferably protected with a cryptographic key, such as a MAC/Signature key. Note that the install does not populate key space, but only allocates key space within an applet.

Card domain **308** manages the post-issuance loading of a new application, while security domain **310** ensures the validity and integrity of the new application once the new application has been loaded onto the smart card. If a security domain **310** is not associated with the newly loaded application, then card domain **308** performs security domain's **310** functions. Once the new application is post-issuance downloaded, various keys such as a cryptographic key and a signature key are preferably utilized for installation and personalization of the application.

APPLET INSTALL COMMANDS EMBODIMENT

This section describes the APDU command response pairs used in the pre issuance and post issuance applet load and install of Open Platform compliant chip cards. The following APDU command response pairs are defined and shall be supported by the Open Platform Card Domain.

Security requirements may vary from one Applet Load/Install to another. The two security alternatives offered in the specification for server authentication by the card shall be supported by the Open Platform Card Domain. At the session level, the server is authenticated once at the beginning of the session (usage of External Authenticate command). At the command level, the server is authenticated at the unitary APDU command (usage of APDU command MACing). Security requirements typically differ between pre issuance and post issuance sessions. For example, pre issuance (identified by the Card Life Cycle Status set to "Masked" or "Initialized") server authentication is at the session level. At post issuance (identified by the Card Life Cycle Status set to "Personalized") server authentication is at the command level. The card authentication by the server is only offered at the session level (usage of Initialized Update command).

The Select command is used for selecting the Card Domain. The Card Domain may be selected either for loading a new applet (or Security Domain) or installing a previously loaded applet (or Security Domain). The Select command message includes the length of the AID and the Card Domain AID. The data field of the response message contains the file control information (FCI) specific to a Card Domain. The FCI returned by a successful selection includes an FCI template, a Card Domain name, proprietary data, applet production life cycle, maximum block length, and FCI discretionary data.

An "applet invalidated" warning condition may be returned by the integrated circuit card (ICC). This is a case where the card is blocked. Blocking the card results in that Card Domain may still be selected with this warning condition, while any other applet present on the card may not be selected any more and shall return the error condition "applet not found". The following error conditions may be returned by the ICC: no precise diagnosis, wrong length, applet not found, incorrect P1 P2.

The Initialize Update command is used to retrieve information data on the integrated circuit card key to be used for the current session and to obtain from the ICC some unique session data. It also initiates the computation by the card of a unique session key and the encryption of a given challenge from the server to the ICC using this session key. The response from the ICC consists of returning to the server some information valid for the session, i.e., key diversification data (card unique data), key information (key and algorithm references), session unique data (ICC challenge to the server), and a cryptogram. The server may verify the cryptogram, i.e. authenticate the ICC, prior to start the load/install process. A previous and successful execution of the Select command is required before processing the Ini-

tialize Update command. A success execution of the Initialize Update command is required before processing an Install command.

The data field of the response message contains the following data elements: key diversification data, key information data, card challenge to server, and cryptogram. The following error conditions may be returned by the ICC: no specific diagnosis, wrong length, conditions of use not satisfied, function not supported (which is the case when the card is blocked), and incorrect P1 P2.

The External Authenticate command asks the currently selected applet in the ICC, here Card Domain, to verify a cryptogram. The cryptogram is computed by encrypting the card challenge to the server returned in the Initialize Update response message using a single DES algorithm in ECB mode with the current session key. A previous and successful execution of the Initialize Update command is required before processing the External Authenticate command. The successful authentication of the server is required prior to the execution of any Install or Load command without Macing. In other words, during pre issuance load/install server authentication is required algorithms.

The External Authenticate command includes in a data field a cryptogram to verify. The key and algorithm references used to compute the cryptogram are known by the ICC un ambiguously, i.e., the session key and the algorithm identified in the Initialize Update response. An "Authentication Failed" warning condition may be returned by the ICC. The following error conditions may be returned by the ICC: no specific diagnosis; wrong length; authentication method blocked; conditions of use not satisfied; or incorrect P1 P2. Installing an applet or Security Do main needs the execution of different functions according to methods invoked by the Card Domain (e.g. install method, check signature). The Install command instructs the Card Domain on which installation step it shall perform. A previous and successful execution of the Initialize Update command is required before processing the Install command. MACing is required for this command when a previous External Authenticate command has not been successfully executed. The Install command message includes a reference control parameter, a security control parameter, length of the data field, data and MAC (if present). The reference control parameter of the Install command is coded to indicate the following meanings: RFU, register, install, load, and authorized. Regarding the register meaning, a future release may support this bit instructing the Card Domain to register a new applet (or Security Domain) into a specific security Domain. In the current release, any installed applet (or Security Domain) is automatically registered in the Card Domain.

Bits 1 to 4 of the reference control parameter control the sequencing of the load/install process. The following bit combinations are valid. The combination 0010 applies to a new applet (or Security Domain). It instructs the Card Domain to accept one or more subsequent Load Applet commands. The Card Domain shall automatically link the new applet or Security Domain (byte code) after the successful execution of the last Load Applet command. It shall not automatically invoke the install method of the Loaded Applet (or Security Domain) after the last Load Applet command. If the original Applet Life Cycle Status is other than "not available" an error condition shall be returned by the ICC.

The bit combination 0011 applies to a new applet (or Security Domain). It instructs the Card Domain to invoke the authorized method of the identified Security Domain with the new applet (or Security Domain) AID. It instructs the Card Domain to accept one or m ore subsequent Lo ad Applet comma nd s only after the authorized method has been successfully executed. The Card Domain should automatically link the new applet (or Security Domain) after the successful execution of the last Load Applet command. It shall not automatically invoke the install method of the loaded applet (or Security Domain) after the last Load Applet command. If the original Applet Life Cycle Status is other than "not available", an error condition is returned by th e ICC.

The bit combination 0100 applies to a previously loaded applet (or Security Domain. It instructs the Card Domain to invoke the install method of the identified applet (or Security Domain) with the install parameters present in the Install command data field. If the Original Life Cycle Status is other than "loaded" an error condition shall be returned by the ICC. A future release will support a combined load and installation of a new applet (or Security Domain).

The security control parameter of the Install command includes the following meanings: RFU; checksum; decrypt; and check signature. Regarding checksum, a future release will support this bit instructing the Card Domain to invoke the checksum method. When loading a new applet (or Security Domain) as indicated in the reference control parameter bits 1 to 3 allow cryptographic controls over the load/install process. The following bit combinations are valid. The bit combination 001 applies to a new applet (or Security Domain). It instructs the Card Domain to invoke the check signature method of the identified Security Domain with the applet (or Security Domain) AID after the successful execution of the last Load Applet command. The bit combination 010 applies to a new or previously Loaded Applet (or Security Domain). It instructs the Card Domain to invoke the decrypt method of the identified Security Domain with the applet (or Security Domain) AID after the successful execution of the last Load Applet command.

The bit combination 011 Applies to a new or previous Loaded Applet (or Security Domain). It instructs the Card Domain to invoke the check signature and decrypt methods of the identified Security Domain with the applet (or Security Domain) AID. The check signature method shall be revoked after the successful execution of the last Load Applet command. The decrypt method shall be automatically revoked only after the check signature method has been successfully executed. The data field of the Install command message contains the following data elements: applet identifier length indicator; AID; Security Domain identifier length indicator; Security Domain identifier; install parameters length indicator; install parameters; and MAC.

The following error conditions may be returned by the ICC: no specific diagnosis, security status not satisfied, conditions of use not satisfied, secure messaging data object missing, secure messaging data object incorrect, incorrect P1 P2, invalid Life Cycle State, authorization failed, applet load failed, signature check failed, decryption failed, checksum failed, installation failed, or registration failed.

This section defines the byte code block structure as transmitted in the load Applet command data field for loading an applet (or Security Domain). The Load Applet command loads one byte code block at the time. Byte code blocks are numbered. The first byte code block shall be numbered 00. The blocked number shall be strictly sequential and incremented by one. The card shall be informed of

the last block. After receiving the last block, the card shall execute the internal process, if any, identified in the security control parameter of the previous Install command. A previous and successful execution of the Install command with its reference control parameter instructing a load is required before processing the Load Applet command. MACing is required for this command when a previous External Authenticgate command has not been successful executed. In other words, MACing is not required during pre issuance load/install.

The load applet command message includes a reference control parameter, block number, length of the block, and byte code block. The data field of the command message contains the byte code block. If a MAC is present, the key and algorithm references used to compute it are known by the ICC unambiguously, i.e., the session key and algorithm identified in the Initialized Update response.

The following error conditions may be returned by the ICC: no specific diagnosis, memory failure, security status not satisfied, conditions of use not satisfied, secure messaging data object missing, secure messaging data object incorrect, not enough memory space, invalid block number, applet load failed, signature check failed, decryption failed, or check sum failed.

APDU INITIALIZATION AND PERSONALIZATION COMMANDS EMBODIMENT

This section describes the APDU command response pairs used for pre issuance and post issuance initialization, personalization and audit of Open Platform compliant chip cards. The following APDU command response pairs are defined: Get Data, Set Status, Select, Initialize Update, External Authenticate, Put Key, Update Record, Update Binary, Put Data, Pin Change/Unblock, and Update Parameter.

Security requirements are applet specific. Two security alternatives are offered in this specification for server authentication by the card. At the session level, the server is authenticated once at the beginning of the session (usage of External Authenticate command). At the command level, the server is authenticated at the unitary APDU command (usage of APDU command MACing). Two security alternatives are also offered in this specification for card authentication by he server. At the section level, the card is authenticated once at the beginning of the session (usage of Initialize Update command). At the command level, the card is authenticated at the unitary APDU command (usage of Update Parameter response MACing). These different security mechanisms may be combined during a session. An applet may also require different security mechanisms during the card life such as a stricter security at post issuance then at pre issuance.

The Get Data command is used to retrieve data objects such as: card production life cycle data, card life cycle status, applet life cycle status, applet personalization key information, or applet production life cycle data of the currently select applet.

The Set Status command is used for incrementing the life cycle status data of the card or the currently selected applets. The state parameter of the Set Status command message includes the following meanings: applet personalized/card load secured and initialized. The reference control parameter of the Set Status command includes the following meanings: card life cycle status, and applet life status. The bit combination 10 for the reference control parameter applies for setting the card cycle life status. It instructs the card domain to increment the state of the card life cycle status. If the Card Domain is not the currently selected environment, an error condition shall be returned by the ICC. In this case, the following combinations for bits of the state parameter of the Set Status command are valid. Bit combination 001 sets card life cycle status from Masked to Initialize. Bit combination 100 sets card life cycle status from Initialized to Load Secured. If the original card life cycle status is other than Masked or Initialized, an error condition shall be returned by the ICC. Blocking the card is ensured using the Card Block command. A blocked card may not be unblocked.

Bit combination 01 of the reference control parameter applies for setting the applet life cycle status. It instructs the currently selected applet to increment its state. If an applet is not the currently selected applet, an error condition shall be returned by the ICC. In this case, the following combination for bits of the state parameter is valid. Bit combination 100 sets Applet Life Cycle Status from Registered to Personalized. If the Original Life Cycle Status is other than Registered, an error condition shall be returned by the ICC. The following state transitions are automatically managed by the Card Domain during the Load/Install process and not by the Set Status command: from Not Available to Loaded, from Loaded, from Loaded to Installed, from Installed to Registered. Blocking an applet is insured using the Applet Block command. Unblocking an applet is insured using the Applet Unblock command. In a future release, the applet installer will allow applet deletion and will manage automatically the state transitions to Not Available.

The following error conditions may be returned by the ICC: no precise diagnosis, memory failure, security status not satisfied, conditions of use not satisfied, secure messaging data object missing, secure messaging data object incorrect, incorrect P1 P2, or invalid life cycle state.

The Select command selects the ICC applet corresponding to the submitted AID, i.e., an Applet, Card Domain, or Security Domain. The response from the ICC consists of returning the Applet, Card Domain or Security Domain file control information (FCI). The data field of the command message contains the AID of the Applet, Card Domain or Security Domain. The data field of the response message contains the FCI specific to the Applet Card Domain or Security Domain. The warning condition "applet invalidated" may be returned by the ICC. This is the case where the applet is present by is blocked. The following error conditions may be return by the ICC: no precise diagnosis, wrong length, function not supported (this is a case where the applet is present but the card is blocked), applet not found (this is a case where the applet does not exist in the ICC or is present but not yet installed) or incorrect P1 P2.

The Initialize Update command is used to retrieve information data on the ICC key to be used for the current session and to obtain from the ICC some unique session data. It may initiate the computation by the card of a unique section key. It may also initiate the encryption of a given challenge from the server to the ICC using -he unique session key. The response from the ICC consists of returning to the server some information valid for the session such as: key diversification data (card unique data), key information (key and algorithm references), session unique data, arid optionally a cryptogram. The server may verify the cryptogram i.e., authenticate the ICC, prior to the start of the personalization process. A previous and successful execution of the Select command is required before processing the Initialize Update command. The reference of the key is known by the ICC unambiguously. Any other value uniquely identifies the key

index within the key set identified in the command message data field. The key index in conjunction with the key set identifier uniquely identifies the key and the algorithm to be used: both references are only applicable to the current context, i.e. the currently selected Applet, Card Domain or Security Domain. When a bit of the reference control parameter is set, the data field of the command message is applet specific and typically contains server session data including some or all of the following data elements: key set identifier, challenge to ICC.

The External Authenticate command will ask the currently selected applet in the ICC to verify a cryptogram. The cryptogram is computed by encrypting the card session unique data returned in the Initialize Update response message with a unique per card applet (or Unique per session) key. A previous and successful execution of the Initialize Update command is required before processing the External Authenticate command. The key and algorithm references used to compute the cryptogram are known by the ICC explicitly, e.g., the ones identified in the initialized update to response. The data field of the command message contains the cryptogram to verify. The warning condition "authentication failed" may be returned by the ICC. The following error conditions may be returned by the ICC: no specific diagnosis, wrong length, authentication blocked, conditions of use no satisfied or incorrect P1 P2.

The Put Key command is used to store or replace a key set (containing one or more keys) or a specific key within a key set with the data provided in the command data field. When already existing in the ICC, the current key set or specific key shall be replaced. A key is identified by a key index within a key set and a key set is identified by a key set identifier: both references are only applicable to the current context, i.e., the currently selected Applet, Card Domain or Security Domain. The value P2 or the Put Key command message shall be set to 00 when more than one key is transmitted to the ICC. P2 shall indicate the key index when a single key is transmitted to the ICC. The key index in conjunction with the reference to the key set of the currently selected applet (or Card Domain or Security Domain) uniquely identifies the key to be stored/replaced.

The reference control parameter indicates if the command data field contains one or more keys. If the command data field contains multiple keys it is coded according to the following structure, including the identification of empty key set entries and the end key set delimiter: a key set identifier is followed by pairs of key indexes, each key index has an algorithm identifier and a key data field.

When the command data field contains only one key (of which the key index is given in P2), it is coded according to the following structure: a key set identifier is followed by an algorithm identifier and a key data field.

Depending on the buffering capability of the ICC for both APDU reception and data decryption, transmitting the keys to the ICC may require one or more command messages. A bit of the reference control parameter indicates if (at least) one subsequent command message shall be expected. When the bit is set, the current APDU is the last (or only) one. When the bit is reset, a subsequent APDU shall be expected and shall refer to the same key index if the current parameter P2 is not set to zero. This case is particular applicable to large keys such as RSA. If parameter P2 of the subsequent APDU is different from the previous one, an error condition shall be return by the ICC.

The command message data field contains (part of) an encrypted key set structure. The entire key set structure (i.e., the key set identifier and key set entries) shall be encrypted. When replacing multiple existing keys, the new key set shall be presented with the same number or keys and the same length for each key as it exists within the card. When replacing a single existing key, the new key shall be presented with the same length as it exists with the card. A future release will support replacement of a key set with a different number of keys and keys with different lengths. If the algorithm identified in the algorithm identifier is not supported, an error condition shall be returned by the ICC. The reference of the encrypting key and algorithm to be used are known implicitly by the ICC according to the current context i.e., the current selected Applet, Card Domain or Security Domain. If a MAC is present, the key and algorithm references used to compute it are known by the ICC implicitly e.g. the ones identified in the initialize update response.

The data field of the response message contains in clear text the key set identifier followed by the key check values presented to the ICC encrypted in the command. These returned key set identifier and key check values may be used by the personalization server to verify the correct decryption by the ICC of the encrypted key set. In other words, this may be used by the server for validating key personalization/update. The following error conditions may be returned by the ICC: no specific diagnosis, memory failure, wrong length, security states not satisfied, conditions of use not satisfied, secure data object missing, secure messaging data object incorrect, incorrect parameters in the data field, not enough memory space, incorrect P1 P2, referenced data not found, algorithm not supported, or invalid key check value. A future release will support ICC internal verification of key check value before key personalization update.

The Update Record command is used to store or replace a record in a linear or cyclic file with the data provided in the command data field. Here it is assumed that before personalization, the Applet (or Security Domain) install process includes both the creation of all the required linear and cyclic files with the appropriate format, number of records, and access conditions and the creation of all the required records of fixed or variable length. If the applet install process only creates the files but not records, the Append Record command shall be used. When already existing in the ICC, the current record shall be replaced. The Update Record command record message includes a record number to be updated, a reference control parameter, a length of the record data, and the record data.

The data field of the command message contains the new record data in clear text. A future release will support personalization of records presented encrypted to the ICC. If the record already exists, the new data is presented in the same format (including the record template tag and length when defined) and with the same length as it exists within the current record in the card. If a MAC is present, the key and algorithm references used to compute it are known by the ICC explicitly, e.g., the ones identified in the initialize update response.

The following error conditions may be returned by the ICC: no specific diagnosis, memory failure, wrong length, command incompatible with file structure, security status not satisfied, condition of use not satisfied, secure messaging data object missing, secure messaging data object incorrect, file not found, record not found, or incorrect P1 P2.

The Append Record command is used to store (but not replace) a single record as a new record at the end of a linear file or at the beginning of a cyclic file with the data provided within the command data field. Here it is assumed that

before personalization, the Applet (or Security Domain) install process includes the creation of all the required linear and cyclic files with the appropriate format, number of records and access conditions but does not include the creation of the records; themselves. If the install process creates all the required records the update record command shall be used. On a successful execution of the command, the ICC increments the record numbering of the file. The first record of a file is numbered 01.

The Append Record message includes a reference control parameter, length of record data and the record data. The data field of the command message contains the new record data in clear text. A future release will support personalization of records presented encrypted to the ICC. The entire record data will be encrypted as a unit. If a MAC is present, the key and algorithm references used to compute it are known by the ICC explicitly, e.g., the ones identified in the Initialize Update response.

The Update Binary command message includes a reference control parameter, a binary offset from the beginning of the file, length of the binary data and the binary data. The data field of the command message contains new binary data in clear text. A future release will support personalization of transparent files presented encrypted to the ICC. The entire sequence of binary data will be encrypted as a unit. If already existing, the new binary data is presented with the same format and length as it exists within the current file in the card. If a MAC is present, the key and algorithm references used to compute it are known by the ICC explicitly, e.g., the ones, identified in the Initialize Update response.

The Put Data command message includes a data object tag. The data field of the command message contains in clear text the new data objects. A future release will support personalization of TLV coded data objects presented encrypted to the ICC. If already existing, the new data objects will be presented with the same format and length as it exists within the card. A future release will support replacement of data objects with different lengths. If a MAC is present, the key and algorithm references used to compute it are known by the ICC explicitly, e.g., the ones identified in the Initialize Update response. The following errors may be returned by the ICC: no specific diagnosis, memory failure, wrong length, security status; not satisfied, conditions of use not satisfied, secure messaging data missing, secure messaging data object incorrect, incorrect parameters in the data field, not enough memory space, incorrect P1 P2, or data object not found.

The PIN Change/Unblock command is used to store or place a PIN value and reset the associated PIN try counter to the value of the PIN try limit within the currently selected applet. When already existing in the ICC, the current PIN shall be replaced. The PIN-Change/Unblock command message includes a reference control parameter, a length of PIN, and the encrypted PIN. The data field of the command message contains the encrypted PIN value. If already existing, the new PIN shall be presented with the same format and length as it exists within the card. A future release will support replacement with a PIN of a different length. When resetting the PIN try-counter, if the PIN try-limit data does not exist, an error condition shall be returned by the ICC. The reference of the encrypting key and the algorithm to be used are known implicitly by the ICC according to the current context, that is typically triple DES algorithm in ECB mode with double length applet personalization (or update) key. If a MAC is present, the key and algorithm references used to compute it are known by the ICC unambiguously, e.g. the ones identified in the Initialize Update response.

The following error conditions may be returned by the ICC; no specific diagnosis, memory failure, wrong length, security status not satisfied, conditions of use not satisfied, secure messaging data object missing, secure messaging data object incorrect, incorrect parameters in the data field, not enough memory space, incorrect P1 P2, or reference data not found.

The Update Parameter command is used to store or replace one or more: tagged data objects provided in the command data field. It also returns to the server a cryptographic proof of its execution. When already existing in the ICC, the current data objects shall be replaced. The reference control parameter of the update parameter command message includes the following meanings; coding according to this specification, clear text data field, and encrypted data field.

The data field of the command message contains a challenge to the ICC from the server followed by the new data objects. When a bit of the reference control parameter is reset, the new data objects are presented to the ICC in clear text. When a bit of the reference control parameter is set, the new data objects are presented to the ICC encrypted. The entire sequence of (one or more) TLV coded data objects, including tags and lengths, will be encrypted as a unit, without any padding of the individual data objects.

Security related command messages (e.g. Initialized Update, Put Key) refer to algorithms and keys in the ICC. The reference may be explicit or implicit. This section defines the key set structure as transmitted in the Put Key command data field. It does not define the ICC internal handling or storage of keys.

A specific key is associated with one and only one algorithm. The algorithm-key pair is defined as a key set entry in a key set. Identifying a key set entry and a key set allows for explicitly referencing an algorithm/key pair (e.g. MAC verification, using triple DES in CBC mode with a double key length). Such referencing prevents from misusing keys or algorithms. A key set is a table algorithm and key pairs valid for the execution of a security related command. A key set may contain one or more algorithm/key pairs. A key set identifier identifies the key set within the current context, i.e. the currently selected applet. In other words, the key set identifier value is relative to the current applet. This implies that a similar value for a key set identifier may be used by different applets. Different key set identifiers are used to differentiate different key versions.

A key set contains a variable number key set entries. A key set entry is an algorithm/key pair valid for the execution of a security related command. A key index identifies the key set entry of the key set identified in the current context, i.e. the currently selected applet. In other words, the key index value is relative to the current key set. This implies that a similar value for a key index may be used by different key sets within a single applet. The first entry of a key set is referred to as the Key Index 01. Different key indexes within one key set may be used to differentiate different key usage and function. Each key set entry is divided into two parts, namely the algorithm identifier and the key data field. The algorithm identifier is coded on one byte according to a table and includes the following meanings; DES-ECB mode, DES-CBC mode, symmetric algorithm, RSA public key, RSA private key, RSA private key-Chinese remainder, and asymmetric algorithms.

A method and system for a smart card domain and a security domain has been disclosed. Software written

according to the present invention may be stored in some form of computer-readable medium, such as memory or CD-ROM, or transmitted over a network, and executed by a processor. Although the present invention has been described in accordance with the embodiment shown, one of ordinary skill in the art will readily recognize that there could be variations to the embodiment and these variations would be within the spirit and scope of the present invention. Accordingly, many modifications may be made by one of ordinary skill in the art without departing from the spirit and scope of the appended claims.

What is claimed is:

1. A method for loading a smart card application onto a smart card after said smart card has been issued to a cardholder, said method comprising:

issuing said smart card to a cardholder, said smart card including a card domain application arranged to manage the post-issuance loading of applications;

establishing communication between said smart card and a provider of said smart card application;

loading said smart card application onto said smart card under control of said card domain application; and

checking a cryptographic signature of said smart card application using said card domain application, whereby said smart card application is loaded onto said smart card post-issuance in a secure manner.

2. A method as recited in claim 1 further comprising:

providing a cryptographic signature key to said card domain application before said smart card application is loaded onto said smart card;

signing said smart card application using said cryptographic signature before said smart card application is loaded onto said smart card, said cryptographic signature being calculated using said cryptographic signature key;

calculating said cryptographic signature of said smart card application once said smart card application has been loaded onto said smart card, said calculating being performed by said card domain application using said cryptographic signature key provided to said card application, whereby said element of checking may be performed by said card domain application.

3. A method as recited in claim 1 further comprising:

decrypting said smart card application by said card domain application once said smart card application has been loaded, whereby said smart card application is loaded onto said smart card post-issuance in a secure manner.

4. A method as recited in claim 3 further comprising:

providing a cryptographic encryption key to said card domain application before said smart card application is loaded onto said smart card;

encrypting said smart card application using said cryptographic encryption key before said smart card application is loaded onto said smart card; and

wherein said element of decrypting being performed by said card domain application uses said cryptographic encryption key provided to said card domain application, whereby said smart card application is loaded onto said smart card post-issuance in a secure manner.

5. A method as recited in claim 1 further comprising:

loading personalization data for said smart card application onto said smart card under control of said card domain application; and

checking a cryptographic signature of said personalization data using said card domain application, whereby said personalization data for said smart card application is loaded onto said smart card post-issuance in a secure manner.

6. A method as recited in claim 1 wherein said smart card further includes a security domain application arranged to manage the security of post-issuance loading of applications, and said element of checking is performed by said security domain application.

7. A method as recited in claim 1 wherein details of said element of checking said cryptographic signature are kept confidential from an issuer of said smart card.

8. A method for loading a smart card application onto a smart card after said smart card has been issued to a cardholder, said method comprising:

issuing said smart card to a cardholder, said smart card including a card domain application arranged to manage the post-issuance loading of applications and a security domain application arranged to manage the security of post-issuance loading of applications;

establishing communication between said smart card and a provider of said smart card application;

loading said smart card application onto said smart card; and

invoking a cryptographic service of said security domain application to validate said smart card application, whereby said smart card application is loaded onto said smart card post-issuance in a secure manner.

9. A method as recited in claim 8 further comprising:

providing a cryptographic signature key to said security domain application before said smart card application is loaded onto said smart card;

signing said smart card application using a cryptographic signature before said smart card application is loaded onto said smart card, said cryptographic signature being calculated using said cryptographic signature key;

calculating said cryptographic signature of said smart card application once said smart card application has been loaded onto said smart card, said calculating being performed by said security domain application using said cryptographic signature key provided to said security domain application, whereby said smart card application is validated.

10. A method as recited in claim 8 further comprising:

decrypting said smart card application by said security domain application once said smart card application has been loaded, whereby said smart card application is validated.

11. A method as recited in claim 10 further comprising:

providing a cryptographic encryption key to said security domain application before said smart card application is loaded onto said smart card;

encrypting said smart card application using said cryptographic encrypt ion key before said smart card application is loaded onto said smart card; and

wherein said element of decrypting being performed by said security domain application uses said cryptographic encryption key provided to said card domain application.

12. A method as recited in claim 8 further comprising:

loading personalization data for said smart card application onto said smart card; and

invoking a cryptographic service of said security domain application to validate said personalization data,

whereby said personalization data for said smart card application is loaded onto said smart card post-issuance in a secure manner.

13. A method as recited in claim 8 wherein details of said element of invoking a cryptographic service are kept confidential from an issuer of said smart card.

14. A method as recited in claim 8 wherein said cryptographic service provides encryption, decryption, MACing, hashing or a digital signature technique.

15. A smart card arranged to load an application onto said smart card after said smart card has been issued to a cardholder, said smart card comprising:

a card domain application arranged to manage loading of said application onto said smart card; and

a security domain application arranged to manage the security of post-issuance loading of applications, said security domain application including

a cryptographic key associated with said application,

a cryptographic service for validating said application after said application has been loaded post-issuance, said cryptographic service using said cryptographic key, and

a key management function associated with said cryptographic key, whereby said application may be loaded onto said smart card post-issuance in a secure manner using said security domain application.

16. A smart card as recited in claim 15 wherein said cryptographic key is an encryption key and wherein said cryptographic service is decryption, whereby said security domain application may decrypt said application after loading of said application onto said smart card.

17. A smart card as recited in claim 15 wherein said cryptographic key is a signature key and wherein said cryptographic service calculates a digital signature, whereby said security domain application may check a signature of said application after loading of said application onto said smart card.

18. A smart card as recited in claim 15 wherein said key management function is loading or updating of a key.

19. A smart card as recited in claim 15 wherein security aspects of said security domain application are kept confidential from said card domain application.

20. A smart card arranged to load a plurality of applications onto said smart card after said smart card has been issued to a cardholder, said smart card comprising:

a card domain application arranged to manage loading of said applications onto said smart card;

a first security domain application arranged to provide security for a first application to be loaded post-issuance, said first security domain application including

a first cryptographic key associated with said first application, said first cryptographic key being kept secret from said card domain and from a second security domain application; and

said second security domain application arranged to provide security for a second application to be loaded post-issuance, said second security domain application including

a second cryptographic key associated with said second application, said second cryptographic key being kept secret from said card domain and from said first security domain application, whereby said first and second applications may be loaded securely post-issuance using said first and second cryptographic keys, respectively.

21. A smart card as recited in claim 20 wherein said first cryptographic key is an encryption key or a signature key and wherein wherein said second cryptographic key is an encryption key or a signature key.

22. A smart card as recited in claim 20 further comprising:

a cryptographic service included with said first security domain for validating said first application after said first application has been loaded post-issuance, said cryptographic service using said first cryptographic key, the details of said cryptographic service being kept secret from said card domain application and from said second security domain application.

23. A smart card as recited in claim 20 further comprising:

a key management function included with said first security domain application.

24. A smart card as recited in claim 23 wherein said key management function is loading or updating of a key.

* * * * *

US006591229B1

(12) **United States Patent**
Pattinson et al.

(10) **Patent No.: US 6,591,229 B1**
(45) **Date of Patent: Jul. 8, 2003**

(54) **METROLOGY DEVICE WITH PROGRAMMABLE SMART CARD**

(75) Inventors: **Neville Pattinson**, Austin, TX (US); **Tibor Somogyi**, Verrieres le Buisson (FR); **Jean-Marc Pietrzyk**, Paris (FR); **Bertrand Du Castel**, Austin, TX (US)

(73) Assignee: **Schlumberger Industries, SA** (FR)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/169,396**

(22) Filed: **Oct. 9, 1998**

(51) **Int. Cl.**[7] **G06F 17/00**

(52) **U.S. Cl.** **702/189**; 702/61; 702/62; 235/376

(58) **Field of Search** .... 702/189, 57, 61, 702/63, 68, 80, 122, 123, 127, 183, 188, FOR 103, FOR 104, FOR 134, FOR 170, FOR 171; 235/382, 492, 380, 375, 376, 485; 705/66, 41, 65, 412, 60, 63; 713/172; 711/115; 717/1–3, 7, 11; 455/558, 557; 340/5.6; 902/26; 725/31; 324/110, 103 R, 113, 116, 140 R, 141, 142

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,977,785	A	8/1976	Harris	355/133
4,168,396	A	9/1979	Best	713/190
4,256,955	A	3/1981	Giraud et al.	235/380
4,278,837	A	7/1981	Best	713/190
4,465,901	A	8/1984	Best	713/190
4,562,306	A	12/1985	Chou et al.	713/200
4,598,810	A	7/1986	Shore et al.	194/205
4,634,807	A	1/1987	Chorley et al.	705/55
4,650,975	A	3/1987	Kitchener	235/375
4,688,169	A	8/1987	Joshi	713/200
4,725,079	A	2/1988	Koza et al.	283/73
4,748,561	A	5/1988	Brown	711/164
4,777,355	A	10/1988	Takahira	235/380
4,797,543	A	1/1989	Watanabe	235/492
4,877,947	A	10/1989	Mori	235/381
4,890,319	A	12/1989	Seth-Smith et al.	380/202
4,890,321	A	12/1989	Seth-Smith et al.	380/231
4,926,480	A	5/1990	Chaum	705/69

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

EP	0 356 237	A2	2/1990
EP	0 380 377	A1	8/1990
EP	0 427 465	A2	5/1991

(List continued on next page.)

OTHER PUBLICATIONS

Translation of EP–0 662 674–A1, pp. 1–15.*
"CyberCash", CyberCash, Inc., Home Page (Internet Feb. 1997), 2 pages.

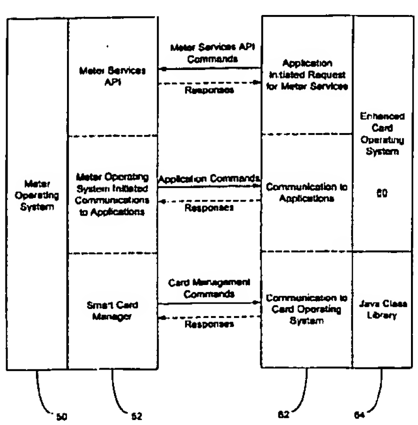
(List continued on next page.)

Primary Examiner—Hal Wachsman
(74) *Attorney, Agent, or Firm*—Straub & Pokotylo; Michael Straub

(57) **ABSTRACT**

A metrology device incorporates a programmable smart card. The smart card may be a Java programmable smart card and allows the metrology device to access Java applications and resources while still retaining independent control over its metrological functions. A smart card interface allows the smart card and metrological device to communicate with each other using the ISO 7816 protocol. The smart card may be external to the metrology device, or it may be fixedly installed in the metrology device. Java applications may be pre-loaded or downloaded on the smart card. The metrological device may select which applications to be run on the smart card. The smart card may have an enhanced operating system that includes various metrological functions as native functions. Alternatively, the metrological functions may be included as part of a Java class library which may be accessed by the operating system or Java applications.

8 Claims, 6 Drawing Sheets



U.S. PATENT DOCUMENTS

4,937,740 A		6/1990	Agarwal et al.	364/200
5,064,999 A		11/1991	Okamoto et al.	235/379
5,068,894 A		11/1991	Hoppe	713/172
5,123,045 A		6/1992	Ostrovsky et al.	713/190
5,195,130 A		3/1993	Weiss et al.	379/98
5,247,575 A		9/1993	Sprague et al.	705/53
5,375,240 A		12/1994	Grundy	713/200
5,386,369 A		1/1995	Christiano	364/464.01
5,406,380 A		4/1995	Teter	358/332
5,412,191 A		5/1995	Baitz et al.	705/5
5,420,405 A		5/1995	Chasek	235/379
5,440,631 A		8/1995	Akiyama et al.	705/53
5,444,779 A		8/1995	Daniele	399/366
5,448,045 A		9/1995	Clark	235/382
5,461,217 A		10/1995	Claus	235/380
5,500,517 A		3/1996	Cagliostro	235/486
5,509,070 A		4/1996	Schull	705/54
5,530,520 A		6/1996	Clearwater	399/366
5,532,920 A		7/1996	Hartrick et al.	707/500
5,537,474 A		7/1996	Brown et al.	380/23
5,544,086 A		8/1996	Davis et al.	364/408
5,550,919 A		8/1996	Kowalski	380/23
5,590,197 A		12/1996	Chen et al.	380/24
5,604,802 A		2/1997	Holloway	380/24
5,613,012 A		3/1997	Hoffman et al.	382/115
5,650,761 A		7/1997	Gomm et al.	235/381
5,689,565 A		11/1997	Spies et al.	380/25
5,692,132 A		11/1997	Hogan	395/227
5,710,706 A		1/1998	Markl et al.	705/409
5,734,150 A		3/1998	Brown et al.	235/381
5,742,756 A		4/1998	Dillaway et al.	395/186
5,761,306 A		6/1998	Lewis	380/21
5,768,419 A		6/1998	Gundlach et al.	395/187.01
5,811,771 A		9/1998	Dethloff	235/380
5,815,657 A		9/1998	Williams et al.	395/186
5,822,517 A		10/1998	Dotan	395/186
5,841,866 A		11/1998	Bruwer et al.	380/23
5,844,218 A		12/1998	Kawan et al.	235/380
5,844,497 A		12/1998	Gray	340/825.34
5,889,941 A		3/1999	Tushie et al.	395/186
5,892,902 A		4/1999	Clark	395/187.01
5,901,303 A		5/1999	Chew	395/400
5,915,226 A		6/1999	Martineau	455/558
5,923,884 A		7/1999	Peyret et al.	395/712
5,937,068 A		8/1999	Audebert	380/23
5,942,738 A	*	8/1999	Cesaire et al.	235/380
6,050,486 A	*	4/2000	French et al.	235/380
6,098,891 A	*	8/2000	Guthery et al.	235/492
6,157,966 A	*	12/2000	Montgomery et al.	235/380
6,170,742 B1	*	1/2001	Yacoob	235/280

FOREIGN PATENT DOCUMENTS

EP	0 513 847 A2		11/1992
EP	0 662 674 A1		7/1995
EP	0 662 674 A1	*	7/1995
EP	0 665 486 A2		8/1995
EP	0 696 121 A1		2/1996
EP	0 717 338 A1		6/1996
EP	0 829 828 A1		3/1998
FR	2 605 770		4/1988
FR	2 667 171		3/1992
FR	2 667 419		4/1992
FR	2 701 133		8/1994
GB	2 191 029 A		12/1987
GB	2 261 973 A		6/1993
JP	95207400		7/1995
WO	WO 96/25724		8/1996
WO	WO 98/09257		3/1998
WO	WO 98/19237		5/1998

OTHER PUBLICATIONS

Phillip M. Hallam-Baker, "Micro Payment Transfer Protocol (MPTP) Version 0.1", W3C Working Draft, Nov. 22, 1995.

"Secure Electronic Transaction (SET) Specification", Book 2: Technical Specifications, Draft for Public Comment, Feb. 23, 1996, pp. 1–10; 129–186.

Blaze, M., "High–Bandwidth Encryption with Low–Bandwidth Smartcards", in *Fast Software Encryption: Third Int'l. Workshop* (ed. D. Gollman), Feb. 1996, pp. 33–40.

"The Copyright Website" Home Page (Internet 7/97), 2 pages.

Intellectual Property: The Property of the Mind, The Economist, Jul. 27, 1996, pp. 57–59.

Lehman, B., "Intellectual Property & the Nat'l. Information Infrastructure", U.S. Patent and Trademark Office, Sep. 1995, pp. 177–200.

"CCC Online", Copyright Clearance Center, Inc. (Internet 7/97), 1 page.

"IBM Cryptolope Containers", IBM infoMarket, International Business Machines Corporation (Internet 8/97), 2 pages.

"IBM infoMarket Rights Management Overview", IBM infoMarket, International Business Machines Corporation (Internet 8/97), 5 pages.

"WIBU Systems Copy Protection", WIBU–Systems AG (Internet 2/97), 1 page.

"Microsoft Authenticode", Microsoft Corporation (Internet 2/97), 1 page.

"McAfee Network Security & Management", McAfee Associates Inc. (Internet 2/97), 2 pages.

"SoftLock Services' Home Page", SoftLock Services, Inc. (Internet 2/97), 1 page.

Bruno Struif, "The Use of Chipcards for Electronic Signatures and Encryption", IEEE Proceedings of the VLSI and Computer Peripherals Conference, May 1989, pp. 4–155 4–158.

Stephen T. Kent, "Protecting Externally Supplied Software in Small Computers", Massachusetts Institute of Technology, Sep. 1980, pp. 2–8, 12–39, 67–76 and 212–236.

R. Mori and M. Kawahara, "Superdistribution: The Concept and the Architecture", The Transactions of the IEICE, vol. E73, No. 7, Jul. 1990, pp. 1133–1146.

John Kelsey and Bruce Schneier, "Authenticating Outputs of Computer Software Using a Cryptographic Co–Processor", Proceedings of CARDIS '96, Amsterdam (1996), (No Month).

Federal Information Processing Standards Publication 190, "Annoucing The Guideline For The Use Of Advanced Authentication Technology Alternatives", Sep. 28,1994, pp. 1–52.

Oded Goldreich and Rafail Ostrovsky, "Software Protection and Simulation on Oblivious RAM's", Journal of the ACM, vol. 43, No. 3, May 1996, pp. 431–473.

Trent Jaeger and Aviel D. Rubin, "Protocols for Authenticated Download to Mobile Information Appliances", The University of Michigan, Dept. of Electrical Engineering & Computer Science, Dec. 1995, pp. 1–12.

"Electronic Payment Schemes", Dr. Phillip M. Hallam-Baker, World Wide Web Consortium (Internet Feb. 1997), 4 pages.

"DigiCash", DigiCash bv, Home Page (Internet Feb. 1997), 2 pages.

"NetChex", Netl, Inc., Home Page (Internet Feb. 1997), 1 page.
"Mondex International", Mondex International Limited, Home Page (Internet Feb. 1997), 1 page.
Wingfield et al., "News: Java Brews Trouble for Microsoft", www.javaworld.com/javaworld, Nov. 1995, pp. 1–2.
Blundon, "The Center of the Universe is a Database", www.javaworld.com/javaworld, Jul. 1996, pp. 1–5.
Gosling, "Audio/Video Sequence of Invited Presentations", www.5conf.inria.fr, May 1996, pp. 1–4.
Sandhu et al., "Authentication, Access Control, and Audit", ACM Computing Surveys, vol. 28, No. 1, Mar. 1996, pp. 241–243.
Cheng et al., "Securing the Internet Protocol", Proc. 5th USENIX UNIX Security Symposium, Salt Lake, Utah, 1995, p. 257. No month.
Kung et al., "Developing an Object–Oriented Software Testing and Maintainance Environment", Communications of the ACM, vol. 38, No. 10, Oct. 1995, pp. 75–87.

J–J. Vandewalle and E. Vetillard, "Developing Smart Card-Based Applications Using Java Card", (Aug. 1998), 18 pages.

F. Thiriet, "Why Java Is Hot for Smart Cards", *ID Systems European Edition*, (Jan. 1998) pp. 32–34.

S. B. Guthery, "Java Card: Internet Computing on a Smart Card", downloaded from: http://computer.org/internet, (Jan.–Feb. 1997), 3 pgs.

* cited by examiner

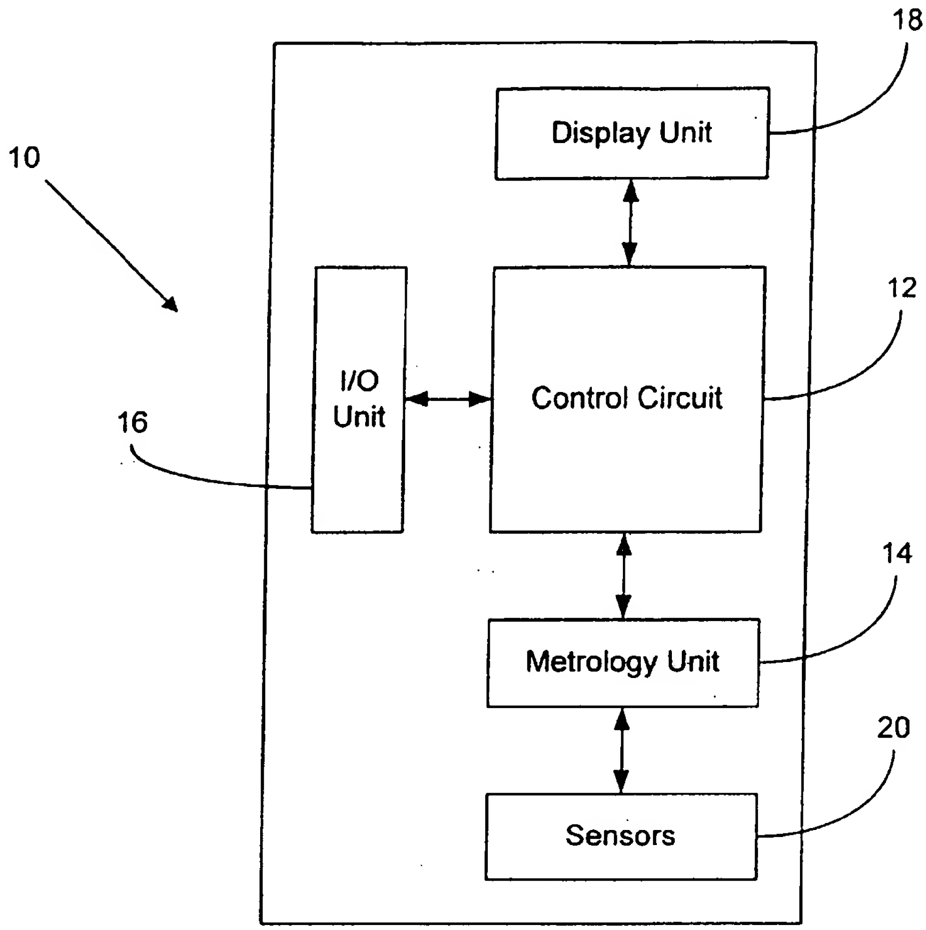


FIG. 1
(PRIOR ART)

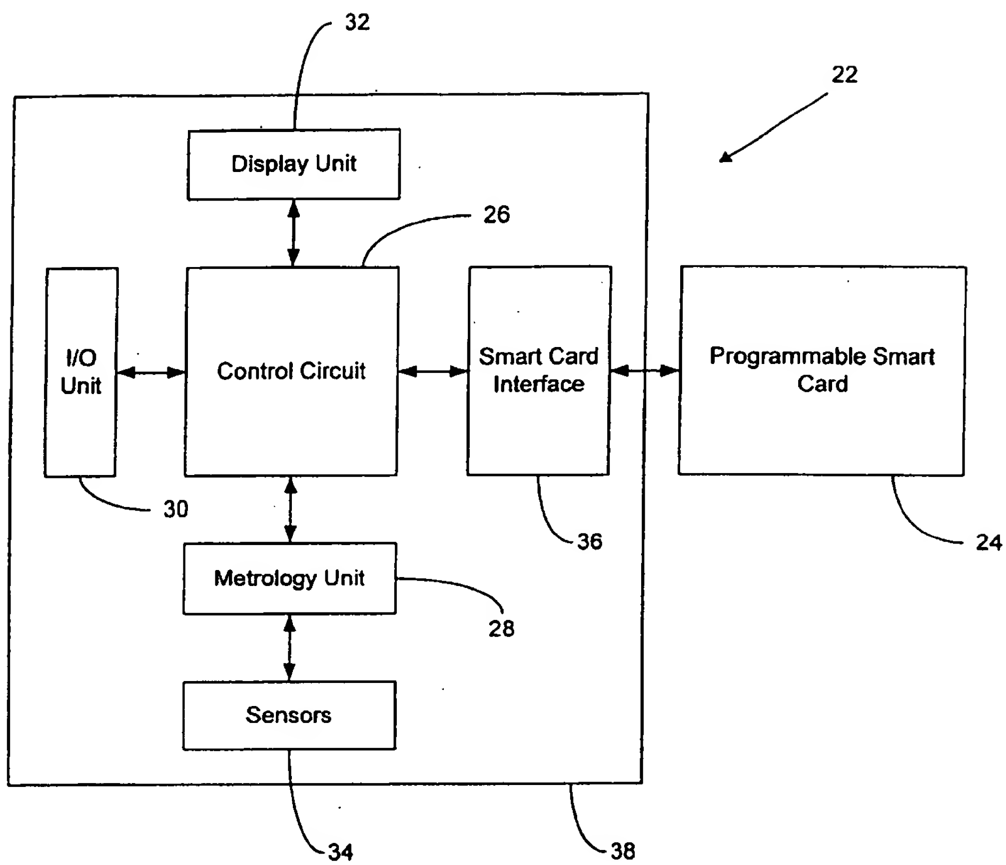

32
22
Display Unit
26
I/O Unit
Control Circuit
Smart Card Interface
Programmable Smart Card
30
36
24
Metrology Unit
28
Sensors
34
38

FIG. 2

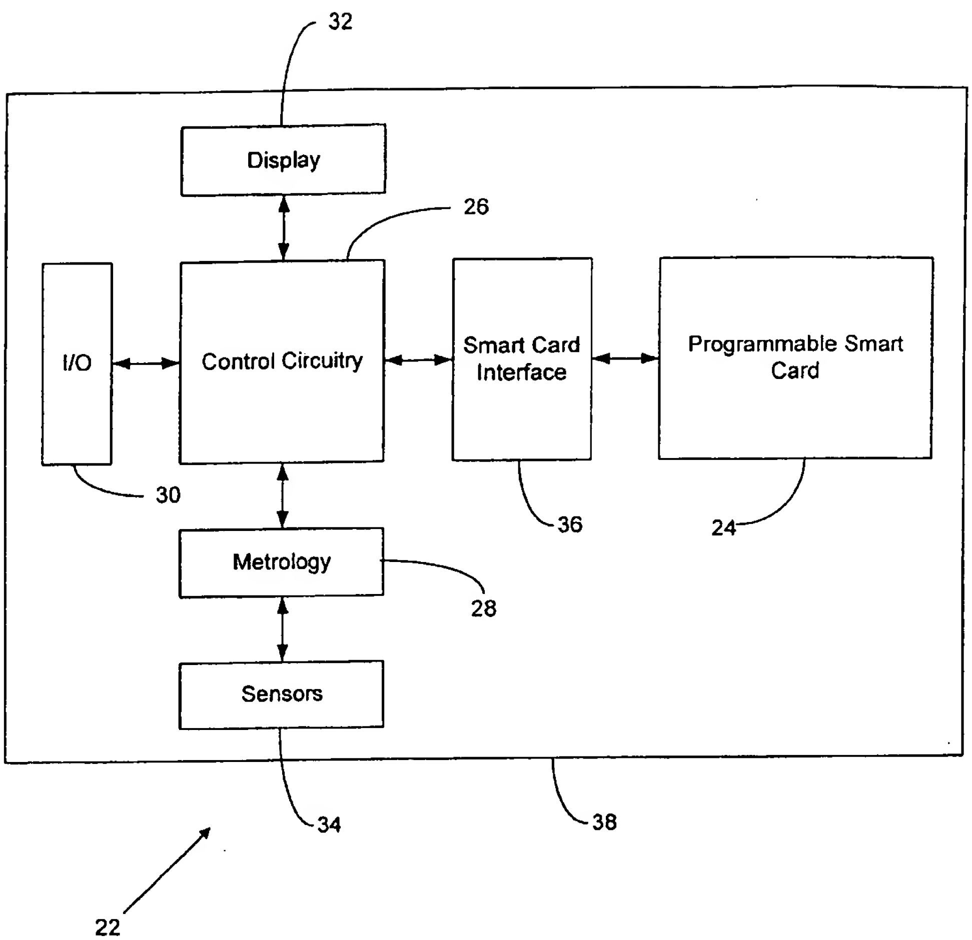

32
Display
26
I/O
Control Circuitry
Smart Card Interface
Programmable Smart Card
30
36
24
Metrology
28
Sensors
34
38
22

FIG. 3

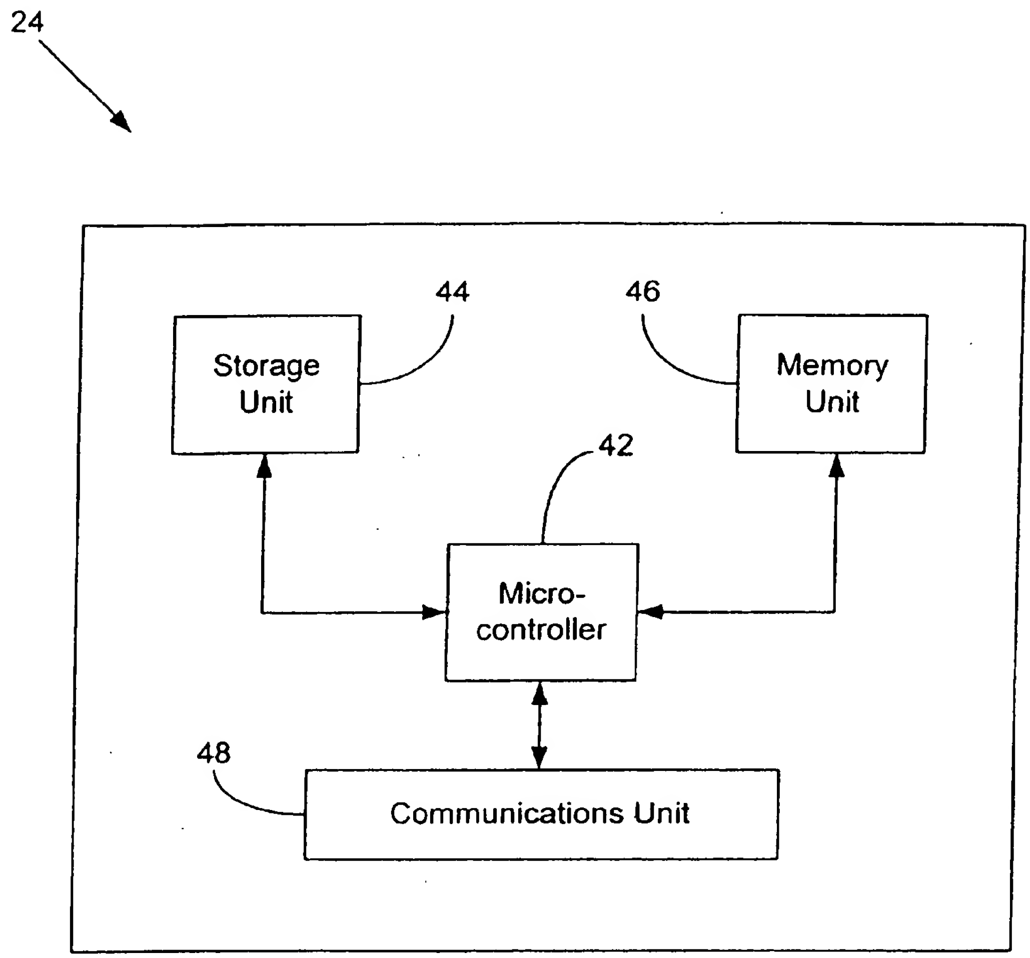
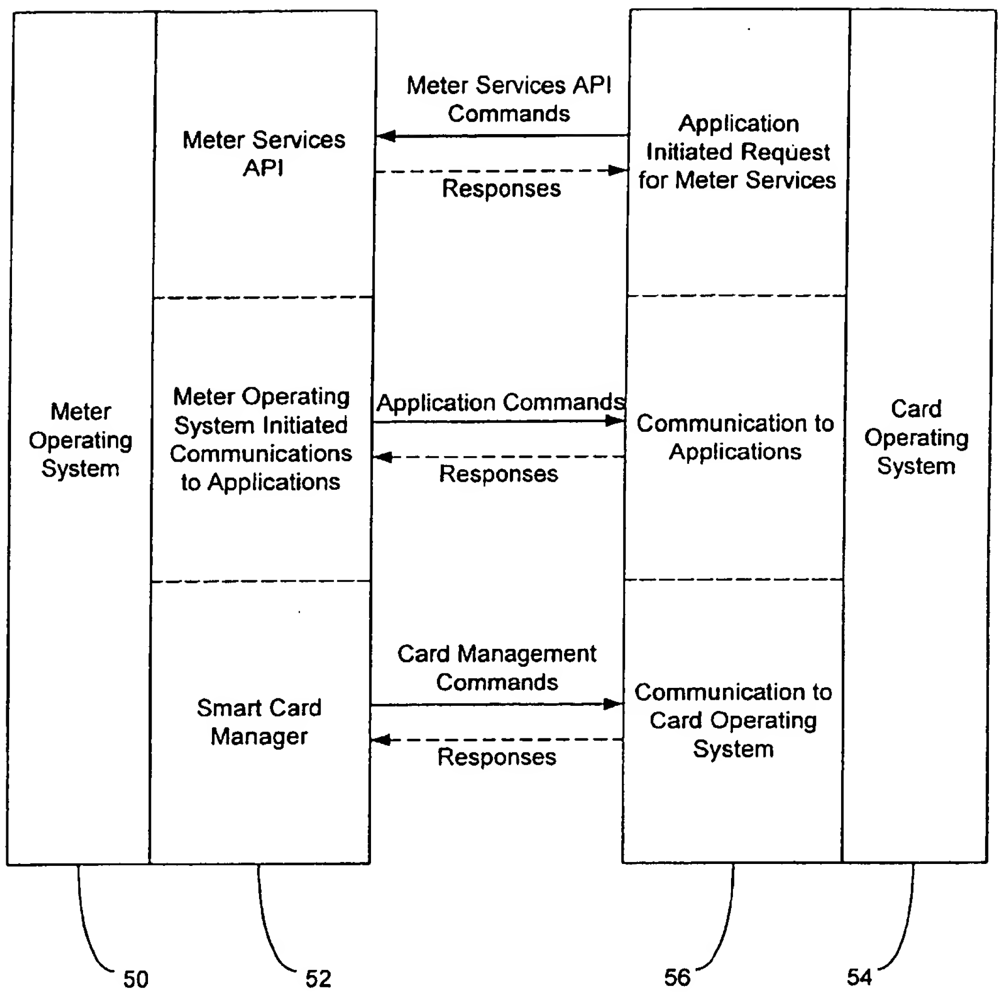

24
44
46
Storage Unit
Memory Unit
42
Micro-controller
48
Communications Unit

FIG. 4



Meter Operating System
Meter Services API
Meter Operating System Initiated Communications to Applications
Smart Card Manager
Meter Services API Commands
Responses
Application Commands
Responses
Card Management Commands
Responses
Application Initiated Request for Meter Services
Communication to Applications
Communication to Card Operating System
Card Operating System
50
52
56
54

FIG. 5

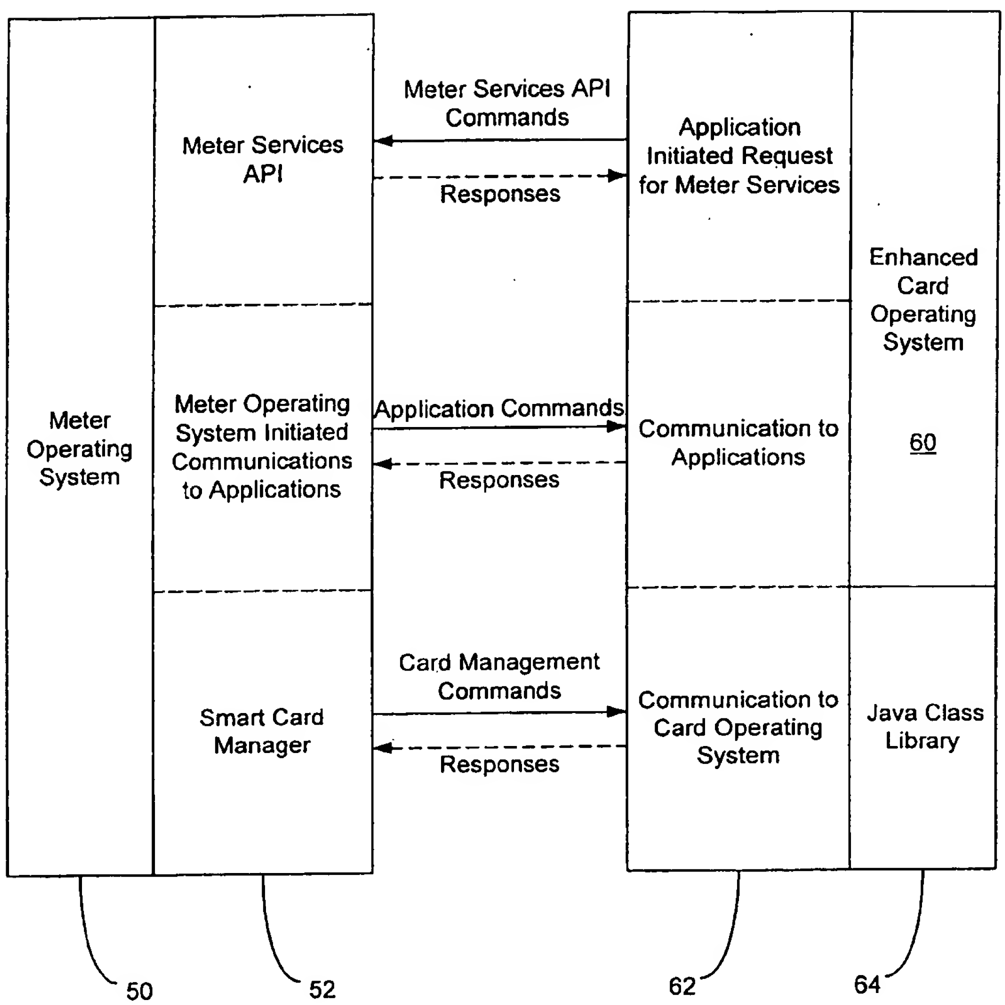

Meter Operating System
Meter Services API
Meter Operating System Initiated Communications to Applications
Smart Card Manager
Meter Services API Commands
Responses
Application Commands
Responses
Card Management Commands
Responses
Application Initiated Request for Meter Services
Communication to Applications
Communication to Card Operating System
Enhanced Card Operating System
60
Java Class Library
50
52
62
64

FIG. 6

METROLOGY DEVICE WITH PROGRAMMABLE SMART CARD

The invention relates to metrology devices and, in particular, to a metrology device with a programmable smart card.

BACKGROUND OF THE INVENTION

Metrology devices, or meters, can be used to measure electricity, water, gas, and other commodities, and can be found in metering applications such as parking meters, payphones, weighing machines, etc. A typical metrology device simply measures a duration, frequency, or amount of a particular commodity and reports what was measured.

Referring to FIG. 1, a prior art meter **10** typically has a central control circuit **12** which is connected to a metrology unit **14**, an I/O unit **16**, and a display unit **18**. The control circuit **12** has a meter operating system running thereon which controls the operation of the meter **10**. The metrology unit **14** is connected to one or more sensors **20** which detect the commodities to be measured, e.g., electricity. The metrology unit **14** measures the commodity detected by the sensor **20** and makes this information available to the control circuit **12**. In some meters, the control circuit **12** actually performs the function of the metrological unit **14** instead of having a separate metrology unit **14** perform the function. The I/O unit **16** typically includes a keypad or buttons and allows a user to input predefined commands to the meter **10**. For example, a user wanting to see how much electricity was consumed last month would simply push the appropriate buttons or otherwise enter the appropriate commands, and the control circuit 12 would retrieve the desired information and display it on the display unit **18**. The display unit **18** may be, for example, an LED, LCD, or other types of displays.

Typically, the accuracy of each meter is tested and certified by an appropriate certification agency before the device is put into use. The certification is good for the entire life of the meter, which is typically around 10 years. Certification requires formal "type"0 testing of any change to a meter's composition or functionality which may affect the meter's metrological function in order to ensure that there are no adverse effects to the meter's accuracy or performance.

Prior art applications have attempted to add functionality to the applications by adding smart cards. For example, Schlumberger's "GSM" mobile telephone products now have a Java programmable smart card in the handsets to identify subscribers and provide information about their service providers. However, it is the GSM "network" that performs the metrological functions and not the GSM handset. A payphone has been developed by Schlumberger Payphones that has a Java Virtual Machine incorporated within the payphone's operating system which allows the payphone to interpret and run Java applications. However, the Java Virtual Machine is then a part of the payphone's operating system as opposed to being a separate and isolated function. Other applications include utility prepayment meters that have removable smart cards which function as transport devices for payment information and allow entry of payment and tariff into the meter. Smart cards, generally, are used for a variety of applications including electronic game cards, bank cards, and identification badges. The smart cards are typically encased in a tamper-resistant, plastic or metal housing about the size of a credit card and contain one or more embedded integrated circuit devices. The functionality of these smart cards, however, are usually predefined at the time they are manufactured.

It would be advantageous to add secure programmability to a meter in order to expand the meter's functionality without interfering with the meter's metrological functions.

SUMMARY OF THE INVENTION

The invention relates to adding a programmable smart card to a metrology device to expand the device's functionality while still maintaining independent operation of the device.

In general, in one aspect, the invention relates to a metrology device comprising a metrology unit, a control circuit connected to the metrology unit, and a smart card interface connected to the control circuit. The control circuit is configured so as to be able to communicate with a programmable smart card through the interface. In one embodiment, the control circuit is configured to communicate with a Java programmable smart card. In another embodiment, the smart card interface is ISO 7816 compliant. In another embodiment, the smart card interface enables full-duplex communication between the smart card and the control circuit. In another embodiment, the control circuit is able to initiate communication with the smart card. In yet another embodiment, the control circuit is configured to send commands to the smart card. In yet another embodiment, the control circuit is configured to execute commands received from the smart card. In yet another embodiment, the control circuit is configured to select an application to be run on the smart card.

In general, in another aspect, the invention relates to a metrology system comprising a programmable smart card, and a metrology device connected to the smart card and configured to communicate with the smart card. In one embodiment, the metrology device comprises a metrology unit, a control circuit connected to the metrology unit, and a smart card interface connected to the control circuit. In another embodiment, the system further comprises a metrology device housing, wherein the smart card is housed within the housing. In yet another embodiment, the smart card is selectively removable from the housing. In yet another embodiment, the system further comprises a meter operating system that controls the operation of the meter, wherein the meter operating system is isolated from a smart card operating system. In yet another embodiment, the smart card is a Java programmable smart card.

In general, in another aspect, the invention relates to a programmable smart card comprising a storage unit configured to persistently store a program to be run on the smart card, a memory unit configured to temporarily store a program to be run on the smart card, and a microcontroller connected to the storage unit and memory unit and configured to selectively execute a metrology related function. In one embodiment, the smart card is a Java programmable smart card. In another embodiment, the microcontroller executes the metrology related function as a native function. In another embodiment, the microcontroller retrieves the metrology related program from a library of available programs.

In general, in another aspect, the invention relates to a method of operating a metrology device with a smart card, the method comprising initiating communication with the smart card, selecting an application to be run on the smart card, and sending commands to the smart card. In one embodiment, the method further comprises receiving commands from the smart card. In another embodiment, the method further comprises receiving the smart card into the metrology device. In yet another embodiment, the method

further comprises performing a metrological function independent of the smart card. In yet another embodiment, the method further comprises providing a result of a metrological function to the smart card. In yet another embodiment, the method further comprises allowing the selected application to run on the smart card independently of the metrology device.

Advantages of the invention include at least the following: the addition of self-contained and secure programmability and functionality to a meter; independent operation of the meter with or without the programmability or functionality; and isolation of the meter's operation system. Other advantages will become apparent from the following description and from the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a prior art metrology device.

FIG. 2 is a block diagram of a metrology device having an external programmable smart card.

FIG. 3 is a block diagram of a metrology device having an internal programmable smart card.

FIG. 4 is a block diagram of a programmable smart card.

FIG. 5 is a block diagram of a metrology device operating system and a smart card operating system.

FIG. 6 is a block diagram of a metrology device operating system and an enhanced smart card operating system.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Throughout the description and the figures, elements that are the same will be accorded the same reference numbers.

Referring to FIG. 2, a metrology device 22 has a programmable smart card 24 connected thereto. In one embodiment, the smart card 24 is a Java programmable smart card such as the Schlumberger Cyberflex smart card. In other embodiments, the smart card 24 may be programmed or may run applications programmed in other programming languages. The metrology device 22 has a control circuit 26 which is connected to a metrology unit 28, an I/O unit 30, and a display unit 32. The metrology unit 28 has one or more sensors 34 connected thereto for detecting a commodity to be measured, e.g., electricity, water, gas, etc. The control circuit 26 may be a microcontroller, microprocessor, ASIC, PAL, or other integrated circuit device. In another embodiment, the control circuit 26 has a meter operating system running thereon which controls the operation of the metrology device 22.

Clm 10

The metrology device 22 further has a smart card interface 36 which connects the metrology device 22 to the smart card 24. The interface 36 provides the necessary physical and electrical connections between the metrology device 22 and the smart card 24 to allow the metrology device 22 and smart card 24 to communicate with each other. In some embodiments, the interface 36 is ISO 7816 compliant and the metrology device 22 and smart card 24 communicate with each other using the ISO 7816 protocol. Although the ISO 7816 protocol provides for half-duplex communication, in some embodiments, other protocols may be used to provide, for example, full-duplex communication between the metrology device 22 and the smart card 24.

A metrology device housing 38 houses the metrology device components described above. The housing 38 may be of any size, shape, or configuration to suit a particular metrology application. In one embodiment, the housing 38 does not house the smart card 24, which is selectively

insertable into and removable from the metrology device 22. In other embodiments, referring to FIG. 3, the smart card 24 may be fixedly installed or otherwise incorporated within the housing 38 as shown. A power supply (not shown), such as a battery or a mains derived power supply, provides power to the metrology device 22, and also to the smart card 24 when connected to the metrology device 22.

In the case where the smart card 24 is insertable into the metrology device 22, the metrology device 22 is provided with means for recognizing the smart card 24 such as an electronic handshake or other means for acknowledging the smart card 24. In a preferred embodiment, such means is compliant with the ISO 7816 protocol.

In operation, the smart card 24 provides a secure, tamper-resistant, and isolated environment within which to perform a variety of functions for the metrology device 22. By way of illustration, the smart card 24 could store cryptographic keys and encode/decode data and/or information for the metrology device 22. In one example, the smart card 24 could validate and authenticate newly downloaded Java applications for the metrology 22 along with providing access to cryptography services for application needing such services. In another example, the smart card 24 could allow the metrology 22 to distinguish between, say, electricity consumption during peak versus off-peak hours, and a different price/rate could be assigned accordingly. One advantage of performing these functions in the smart card 24 instead of the metrology device 22 is the metrology device 22 may be susceptible to probing or tampering, or its security otherwise compromised, whereas the smart card 24 is secure and tamper-resistant.

Referring to FIG. 4, the programmable smart card 24 comprises a microcontroller 42 which is connected to a storage unit 44 and a memory unit 46. The microcontroller 42 executes smart card software and programs, carries out meter instructions, and generally manages the flow of data to and from the smart card 24. In some embodiments, the microcontroller 42 may include a microprocessor, a programmable array logic (PAL), an application-specific integrated circuit (ASIC), and/or other integrated circuit devices. The storage unit 44, which may include a read-only memory (ROM), stores the programs and data that are needed to operate the smart card 24. The memory unit 46, which may include a random-access-memory (RAM), temporarily stores the programs and data used by the microcontroller 42 during program execution. New or updated programs, applications, or data may be downloaded or programmed into the smart card 24 from time to time to upgrade the smart card 24. Also, smart cards containing new or updated programs, applications, or data may be mailed to the desired locations and then inserted into a metrology device. The smart card 24 also has a communications unit 48 connected thereto which allows the microcontroller 42 to transfer data to and from external devices.

In another embodiment, in addition to a physical interface, the meter also has a software interface which isolates the operating system of the meter from the operating system of the smart card. Referring to FIG. 5, a meter has an operating system 50 and a defined interface 52. The meter interface 52 functions primarily to 1) manage the smart card, 2) initiate communications with Java applications, and 3) respond to requests for meter services by the Java application. Similarly, a smart card includes an operating system 54 and a smart card software interface 56 that functions primarily to: receive managerial or administrative commands and pass those commands onto the smart card operating system 54, receive Java application commands and pass

those commands onto the Java applications, and send meter services requests from the Java applications to the meter.

In operation, for example, the meter operating system **50** may issue a managerial or administrative command to the smart card, such as a command to load a particular application. The meter interface **52** converts or otherwise changes the command to comply with the ISO 7816 protocol or other suitable protocols, and sends the command to the smart card (as shown by the solid line arrow). The smart card interface **56** receives the managerial or administrative command and passes it to the smart card operating system **54** which may then acknowledge the command or otherwise respond to the command (as shown by the dashed line arrow). The meter interface **52** also allows the meter operating system **50** to initiate communication with a selected application, for example, a Java application on the smart card and to instruct the application to perform one or more specific tasks. The smart card interface **56** receives the instruction and passes it to the appropriate application which may then acknowledge or otherwise respond to the instruction. Once an application is selected and activated, the application may call on the meter to provide a certain metrological service. The application may simply issue a request for that service, and the smart card interface **56** then converts the request into the appropriate protocol and sends the request to the meter. The meter interface **52** receives the metrological service request and passes it to the meter operating system **50**. The meter operating system **50** determines whether the metrological service is available and causes the appropriate service to be performed. The results of the service are then sent to the application through the meter interface **52**.

The software meter interface **52** allows the meter to access the smart card's programmability, applications, and resources while still allowing the meter to carry out its metrological functions independently of the smart card. The meter is able to operate normally with or without the smart card, and the smart card's programmability and functionality become available to the meter only when the smart card is inserted, installed, or otherwise connected to the meter. This arrangement has the advantage in that any functionality introduced into the meter by the applications can be easily proven (type tested) to not affect the accuracy of the metrological functions of the meter, thereby not compromising the meter's certification.

Similarly, the ability of the smart card to download and run applications independent of the meter is not affected. The applications could also be pre-loaded on the smart card prior to insertion or installation in the meter.

In yet another embodiment, referring to FIG. **6**, a smart card operating system **60**, for, for example, a Java programmable smart card, may have various metrological functions built-in to the operating system **60**. For example, if one or more metrology related mathematical calculations (e.g., average daily use) are repeatedly performed by one or more Java applications, the calculations may be incorporated into the smart card operating system **60** as native functions of the operating system **60**. The functions may then be available to all Java applications and may be run directly from the operating system **60** instead of in the Java application which requires a Java Virtual Machine to interpret the application. This arrangement has the advantage of being much faster because the functions are executed rather than interpreted. Also, the functions may require less storage space as a part of the operating system **60** compared to a Java application, although the size of the operating system **60** may increase. In an alternative embodiment, the functions could be implemented as a part of a Java class library **64** which may then be made available to all applications.

It is to be understood that the embodiments described herein are illustrative only, and that other embodiments may be derived by one of ordinary skill in the art without departing from the scope of the invention. For example, referring to FIG. **2**, the control circuit **26**, metrology unit **28**, I/O unit **30**, display unit **32**, sensors **34**, and smart card interface **36** may all be combined into a single integrated circuit device, or an otherwise smaller or larger number of integrated circuit devices.

What is claimed is:

1. A utility metrology device, comprising:

a metrology unit for metering usage of a commodity selected from the set having the members electricity, gas, and water;

a control circuit connected to the metrology unit;

a smart card interface connected to the control circuit,

wherein the control circuit is configured to:

i) communicate with a programmable smart card through the smart card interface;

ii) send commands to the programmable smart card;

iii) select an application to be run on the programmable smart card;

iv) instruct said application to perform a specific task;

v) execute a command from the programmable smart card; and

vi) perform the appropriate service corresponding to said command from the programmable smart card; and

a meter operating system that controls the operation of the utility metrology device, said meter operating system being isolated from a smart card operating system.

2. The utility metrology device of claim **1**, wherein the control circuit is configured to communicate with a programmable smart card that is programmable in a high level language.

3. The utility metrology device of claim **1**, wherein the smart card interface enables full-duplex communication between the programmable smart card and the control circuit.

4. The utility metrology device of claim **1**, wherein the control circuit initiates communications with the programmable smart card.

5. A utility metrology system which may be reprogrammed comprising:

a metrology device including:

a metrology unit for metering usage of a commodity selected from the set having the members electricity, gas and water;

a control circuit connected to the metrology unit; and

a smart card interface connected to the control circuit,

wherein the control circuit is configured to:

i) communicate with a programmable smart card through the smart card interface;

ii) send a command to the programmable smart card;

iii) select an application to be run on the programmable smart card;

iv) instruct said application to perform a specific task;

v) execute a command from the programmable smart card; and

vi) perform the appropriate service corresponding to said command from the programmable smart card; and

a meter operating system that controls the operation of the device, said meter operating system being isolated from a smart card operating system; and

a programmable smart card connected to the metrology device and operable to communicate therewith via said smart card interface, said programmable smart card including said smart card operating system.

6. The utility metrology system of claim 5, further comprising a metrology device housing, wherein the programmable smart card is housed within the metrology device housing.

7. The utility metrology system of claim 6, wherein the programmable smart card is selectively removable from the metrology device housing.

8. The utility metrology system of claim 5, wherein the programmable smart card is a smart card programmable in a high level language.

* * * * *

US006216227B1

(12) **United States Patent**
Goldstein et al.

(10) **Patent No.: US 6,216,227 B1**
(45) **Date of Patent: Apr. 10, 2001**

(54) **MULTI-VENUE TICKETING USING SMART CARDS**

(75) Inventors: **Theodore Charles Goldstein**, Palo Alto; **Jonathan B. Ziegler**, Cupertino, both of CA (US)

(73) Assignee: **Sun Microsystems, Inc.**, Palo Alto, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/106,600**

(22) Filed: **Jun. 29, 1998**

(51) **Int. Cl.**^{7} **H04L 9/32**
(52) **U.S. Cl.** **713/172**; 705/65; 705/67
(58) **Field of Search** 705/65, 67; 713/172

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,598,477	1/1997	Berson	380/51
5,721,781 *	2/1998	Deo et al.	705/67
5,754,654 *	5/1998	Hiroya et al.	705/76
6,005,942 *	12/1999	Chan et al.	713/187

FOREIGN PATENT DOCUMENTS

628928 A1	12/1994	(EP)	G07B/15/00
658862 A2	6/1995	(EP)	G07F/7/10
658862 A3	6/1995	(EP)	G07F/7/10
823694 A1	2/1998	(EP)	G07F/7/08
829828 A1	3/1998	(EP)	G07F/7/08

OTHER PUBLICATIONS

Rinaldo Di Giorgio, iButtons: The first ready–to–buy 2.0 Java Card API devices, Apr. 1998, 9 pages, Java World.
Wolfgang Rankl, et al., "Handbuch der Chipkarten," (in German), 1995, pp. 64–65, 202–205. English translation also provided.

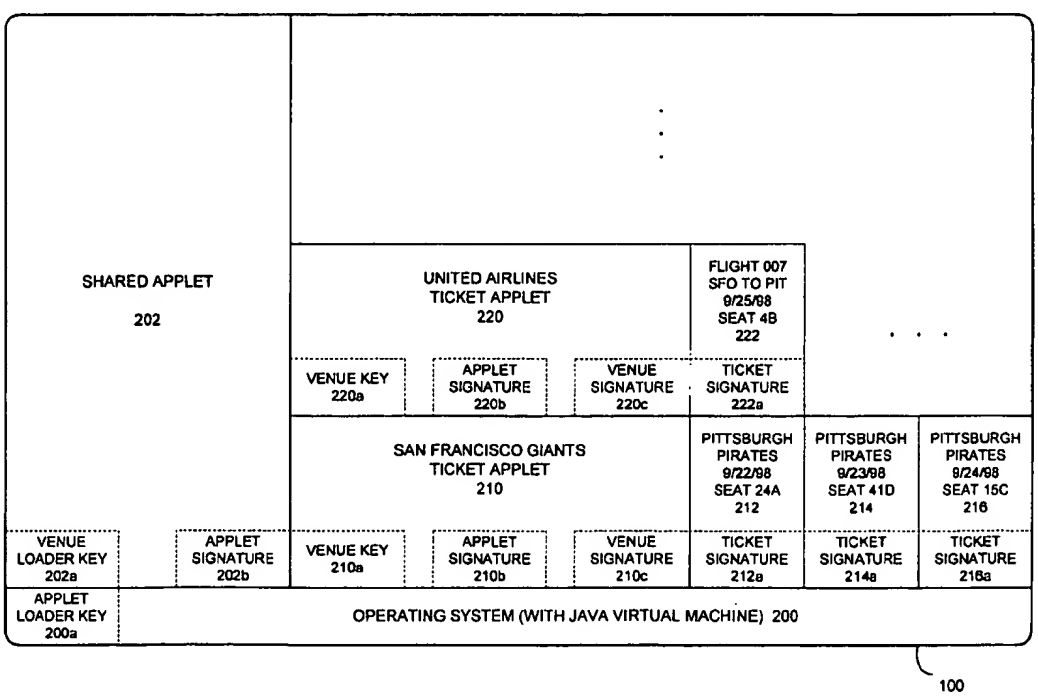
* cited by examiner

Primary Examiner—Gilberto Barron, Jr.
(74) *Attorney, Agent, or Firm*—Park & Vaughan LLP

(57) **ABSTRACT**

A system and methods are provided for storing and validating electronic tickets for multiple venues on a single smart card. In accordance with this present embodiment, an operating system of the smart card includes a Java Virtual Machine and an applet loader key. A shared applet, including a venue loader key, is validated with the applet loader key and is stored on the smart card. One or more venue applets are also stored on the smart card, each with a venue key corresponding to an associated venue. Each venue applet is validated by the applet loader key and the venue loader key. The shared applet is used by the venue applets to interface with ticket loaders and ticket validation devices. Tickets are purchased for events associated with the venue applets and are stored on the smart card in association with the related venue applets. Ticket signatures are authenticated with each venue applet's venue key. A ticket is cancelled after being tendered to gain admittance to an event.

25 Claims, 5 Drawing Sheets



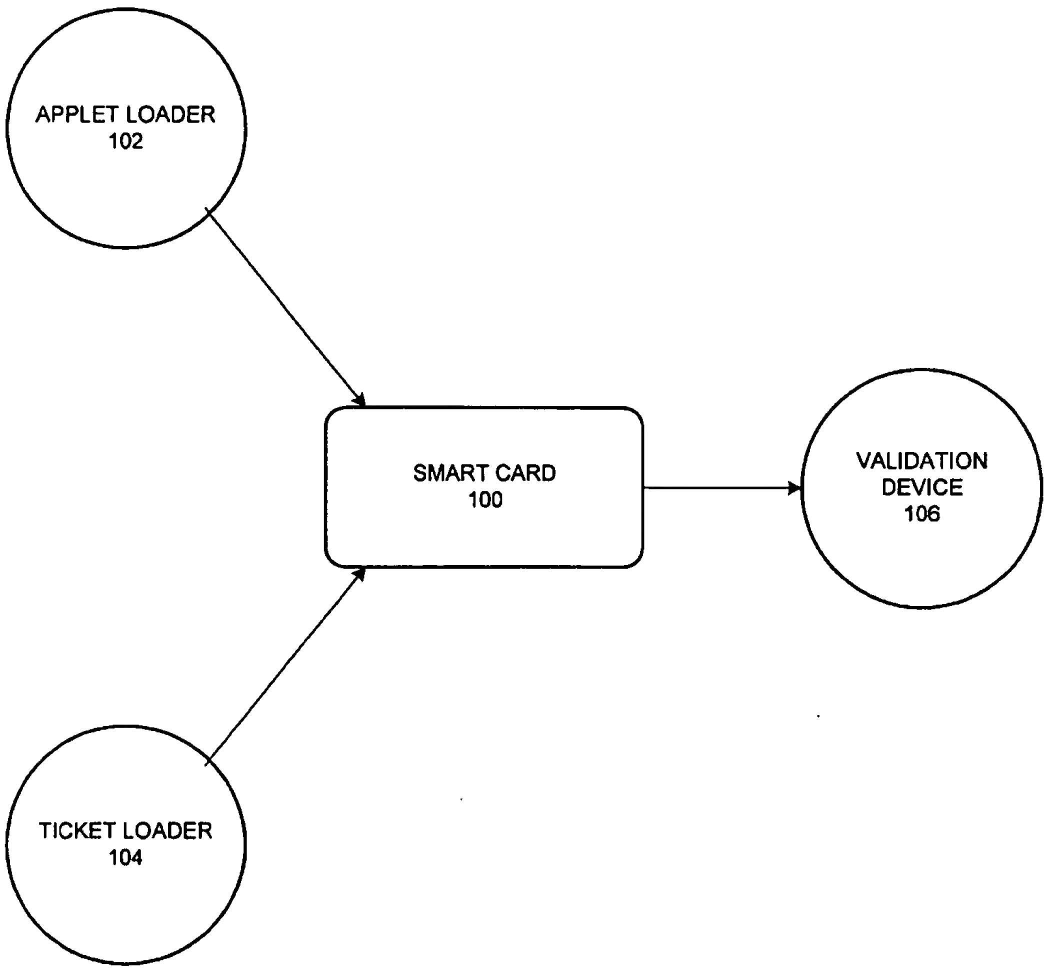

APPLET LOADER
102
SMART CARD
100
VALIDATION
DEVICE
106
TICKET LOADER
104

FIG. 1

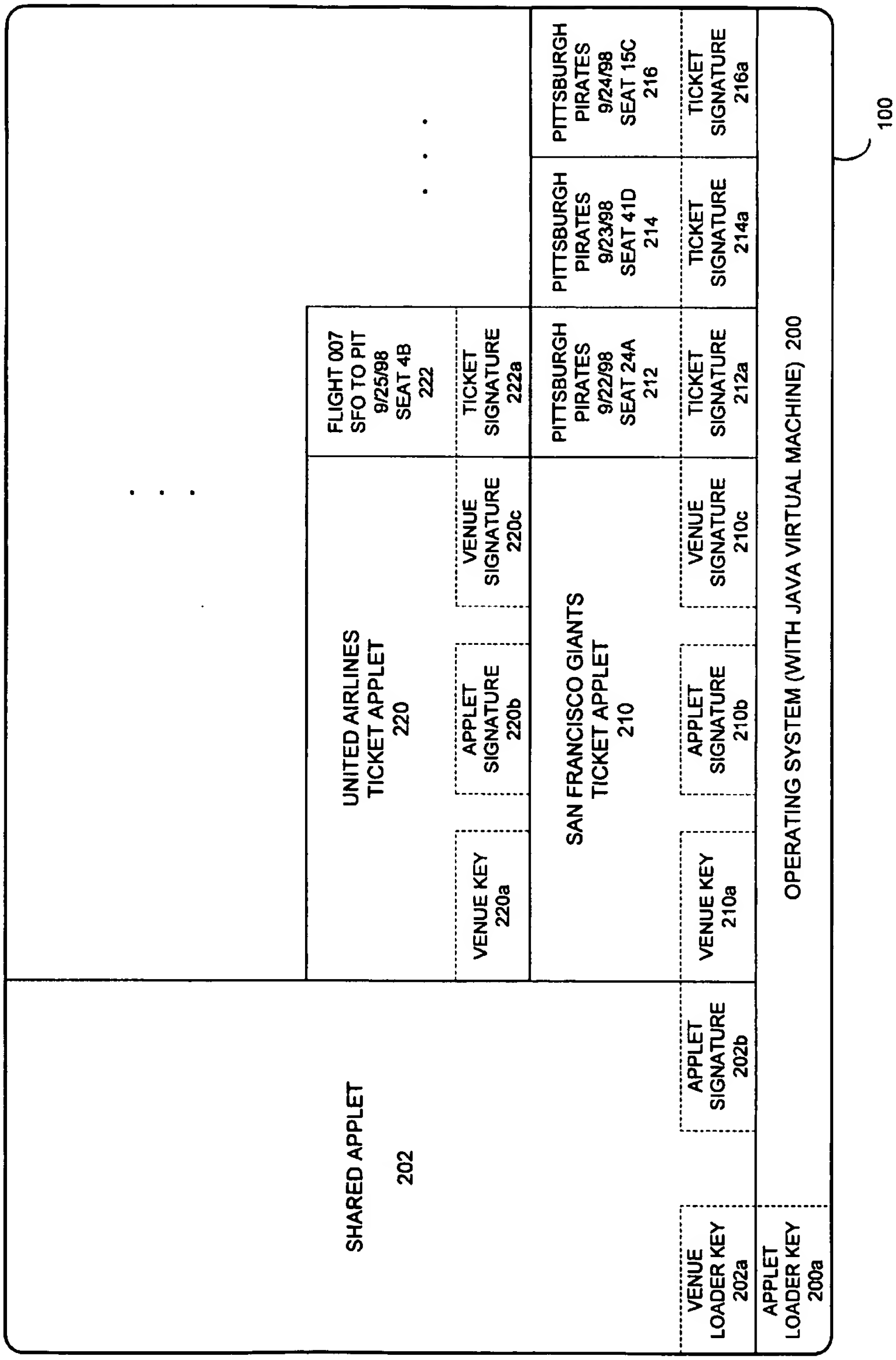

FLIGHT 007
SFO TO PIT
9/25/98
SEAT 4B
222
TICKET SIGNATURE 222a
PITTSBURGH PIRATES 9/22/98 SEAT 24A 212
PITTSBURGH PIRATES 9/23/98 SEAT 41D 214
PITTSBURGH PIRATES 9/24/98 SEAT 15C 216
TICKET SIGNATURE 212a
TICKET SIGNATURE 214a
TICKET SIGNATURE 216a
UNITED AIRLINES TICKET APPLET 220
VENUE KEY 220a
APPLET SIGNATURE 220b
VENUE SIGNATURE 220c
SAN FRANCISCO GIANTS TICKET APPLET 210
VENUE KEY 210a
APPLET SIGNATURE 210b
VENUE SIGNATURE 210c
SHARED APPLET 202
VENUE LOADER KEY 202a
APPLET SIGNATURE 202b
APPLET LOADER KEY 200a
OPERATING SYSTEM (WITH JAVA VIRTUAL MACHINE) 200
100

FIG. 2

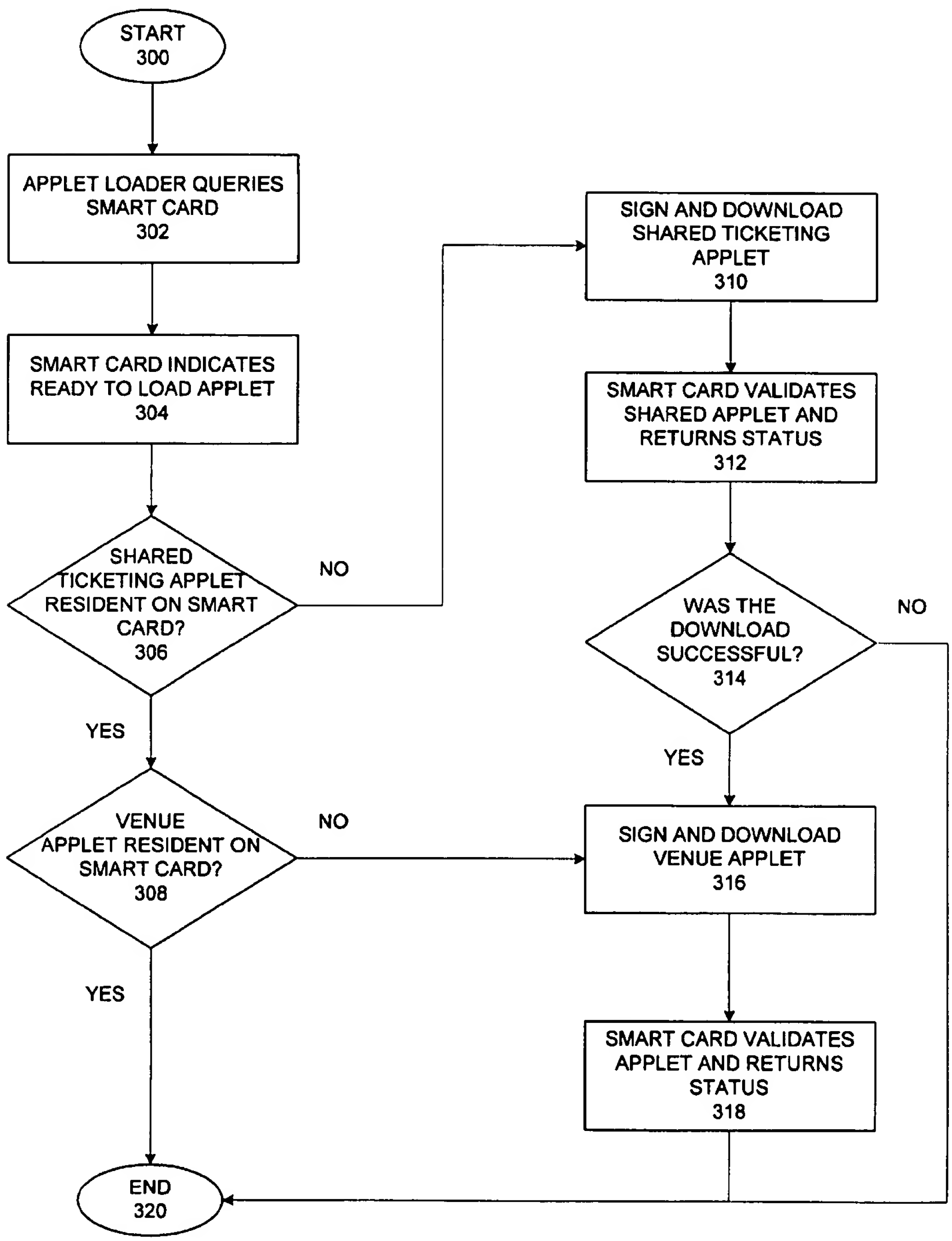

START
300
APPLET LOADER QUERIES SMART CARD
302
SMART CARD INDICATES READY TO LOAD APPLET
304
SHARED TICKETING APPLET RESIDENT ON SMART CARD?
306
NO
YES
VENUE APPLET RESIDENT ON SMART CARD?
308
NO
YES
SIGN AND DOWNLOAD SHARED TICKETING APPLET
310
SMART CARD VALIDATES SHARED APPLET AND RETURNS STATUS
312
WAS THE DOWNLOAD SUCCESSFUL?
314
NO
YES
SIGN AND DOWNLOAD VENUE APPLET
316
SMART CARD VALIDATES APPLET AND RETURNS STATUS
318
END
320

FIG. 3

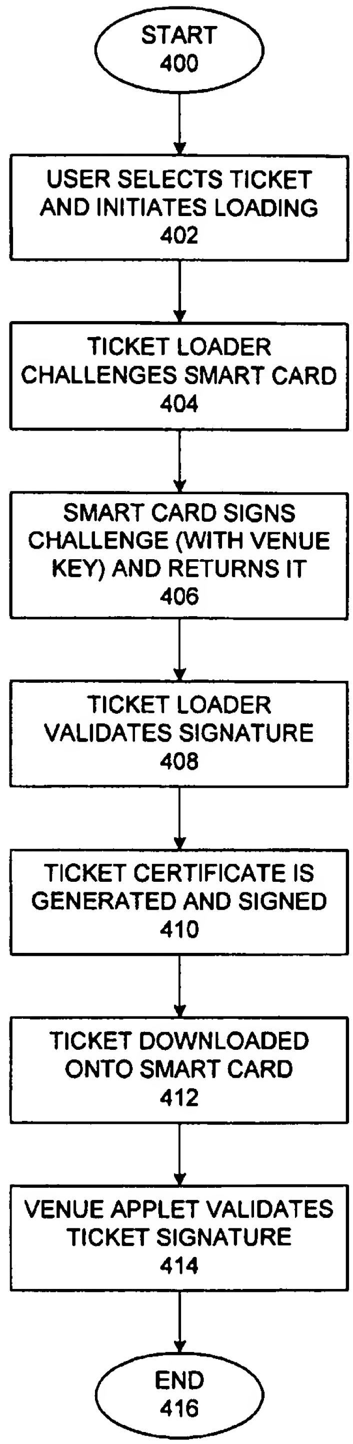

START
400
USER SELECTS TICKET AND INITIATES LOADING
402
TICKET LOADER CHALLENGES SMART CARD
404
SMART CARD SIGNS CHALLENGE (WITH VENUE KEY) AND RETURNS IT
406
TICKET LOADER VALIDATES SIGNATURE
408
TICKET CERTIFICATE IS GENERATED AND SIGNED
410
TICKET DOWNLOADED ONTO SMART CARD
412
VENUE APPLET VALIDATES TICKET SIGNATURE
414
END
416

FIG. 4

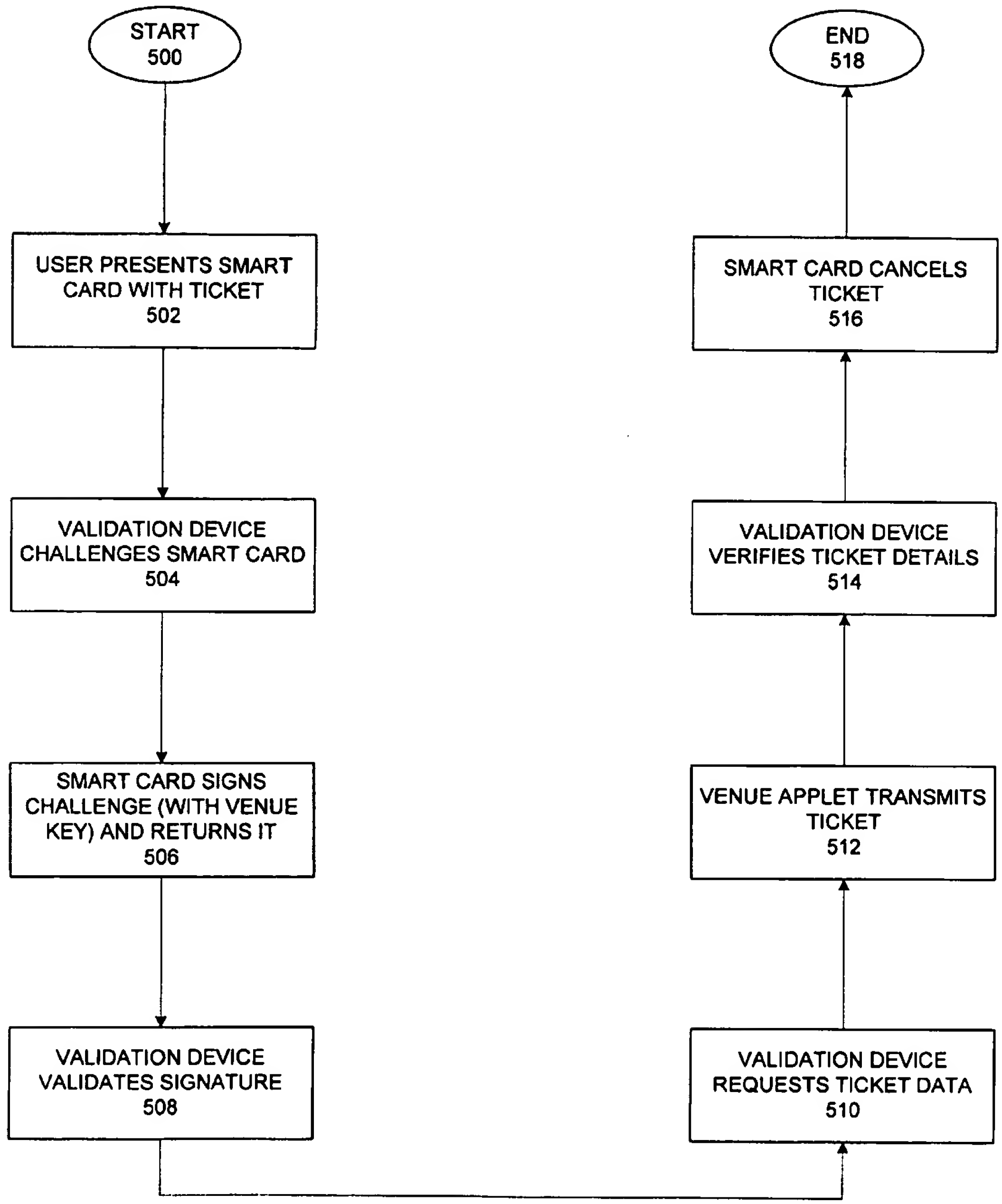

START
500
USER PRESENTS SMART CARD WITH TICKET
502
VALIDATION DEVICE CHALLENGES SMART CARD
504
SMART CARD SIGNS CHALLENGE (WITH VENUE KEY) AND RETURNS IT
506
VALIDATION DEVICE VALIDATES SIGNATURE
508
VALIDATION DEVICE REQUESTS TICKET DATA
510
VENUE APPLET TRANSMITS TICKET
512
VALIDATION DEVICE VERIFIES TICKET DETAILS
514
SMART CARD CANCELS TICKET
516
END
518

FIG. 5

MULTI-VENUE TICKETING USING SMART CARDS

BACKGROUND

This invention relates to the field of electronic commerce. More particularly, a system and methods are provided for electronic ticketing.

The use of tickets for sporting venues, entertainment events, travel and the like is no longer strictly a mechanical function. Ticketing systems have evolved to make use of computer systems in various phases of the ticket generation, issuance and validation processes.

For example, in U.S. Pat. No. 5,598,477, issued to Berson, a customer submits information concerning a desired ticket (e.g., scheduling data pertaining to an airline flight). A data processing system sends ticketing information and encrypted validation data to a local printing system. The local system prints the ticket, which includes the validating information encoded in a two-dimensional barcode. The customer presents the ticket at flight time, where a validating system scans the barcode, transforms the data from physical form into digital form and validates it. If valid, the customer receives his boarding pass, luggage claim checks, etc.

Berson, however, still requires the issuance of a paper ticket. Paper tickets are, of course, subject to theft, mutilation, destruction, loss, etc. In addition, a ticket produced according to the Berson system is necessarily good for only onetime use. The ticket is physically collected at the time of the flight. Two additional disadvantages exist with this scheme. First, the use of two-dimensional barcodes requires printers capable of producing, and barcode scanners capable of reading, such barcodes. Depending upon the number of sites at which tickets are printed or accepted, this may involve significant cost. Second, the use of cryptographic means to secure the validation information requires a sophisticated key management scheme.

In a modification of the Berson system, large random numbers may be used in place of cryptographic security. A particular random number is chosen and printed as a one-dimensional barcode on a physical ticket. The use of large numbers significantly decreases the chance of a person correctly guessing the number assigned to a particular ticket for a discrete event (e.g., airplane flight, entertainment event). The random numbers are stored in a database accessible to sites at which the tickets are used. When the ticket is presented at a site, the number on the ticket is compared to the list of valid numbers stored in the database. This scheme still possesses the disadvantages inherent in paper tickets, such as destruction or mutilation and the limitation to a single use. In addition, without further protection, the database of random numbers provides a single point of vulnerability. A person with access to the database could conceivably generate large quantities of bogus tickets,

In addition to the above disadvantages, known ticketing systems provide admission to only a single event or a single site. Also, a paper ticket issued by a known system is not generally modifiable without physically replacing the issued ticket. In other words, a person who wishes to visit or enjoy multiple events or multiple venues must carry and present a different ticket for each event or venue. As he or she makes plans to visit even more events or venues, additional paper tickets must be purchased and carried, thus increasing the risk of loss.

SUMMARY

In one embodiment of the invention, a system and methods are provided for storing, on a single electronic device (e.g., smart card, hand-held computer), electronic tickets to events offered at multiple venues. In this embodiment, the electronic device receives and stores a venue module associated with each venue for which a ticket is purchased. The venue module enables the electronic device to store tickets for the associated venue, and includes a venue key for validating individual tickets. The electronic device also receives and stores a shared ticketing module containing instructions to be called by one or more venue modules. The shared ticketing module includes a "venue loader key" for validating installed venue modules.

After the electronic device is configured with the shared ticketing module and one or more venue modules, tickets for each installed venue module may be stored. In a present embodiment of the invention, the electronic device's user identifies parameters (e.g., event, date, time, seat) for a ticket and the corresponding electronic ticket is downloaded from a ticket loader, along with a ticket signature. The venue module for the corresponding venue module authenticates each stored ticket's signature using its venue key.

When a ticket is to be presented for admission to an event, in a present embodiment a validation device challenges the electronic device by issuing a challenge code. The venue module for the event's venue signs the code with its venue key and returns the signed code. After the signature is validated, the electronic device transmits the ticket for the event and the ticket is canceled.

BRIEF DESCRIPTION OF THE FIGURES

FIG. 1 is a block diagram depicting one system in which a smart card is used to store venue applets and tickets for admission to a venue in accordance with an embodiment of the present invention.

FIG. 2 depicts a smart card populated with multiple venue applets and tickets in accordance with an embodiment of the present invention.

FIG. 3 is a flow chart demonstrating one method of loading a venue applet onto a smart card in accordance with an embodiment of the present invention.

FIG. 4 is a flow chart demonstrating one method of loading a ticket onto a smart card in accordance with an embodiment of the present invention.

FIG. 5 is a flow chart demonstrating one method of validating a ticket stored on a smart card in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION

The following description is presented to enable any person skilled in the art to make and use the invention, and is provided in the context of a particular application and its requirements. The present invention is not intended to be limited to the embodiments shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.

For example, in a present embodiment of the invention, cryptographic means are applied to ensure the security of electronic tickets and venue modules, or applets (e.g., small Java applications), that are loaded onto smart cards. One skilled in the art will recognize that the purpose of the cryptographic keys described below is to ensure the security and authenticity of information stored on a smart card, and

does not necessarily rely upon a particular cryptographic scheme unless otherwise indicated. Various cryptographic keys are therefore described below for various purposes. The invention is not limited to a particular method of cryptographic security, however, and specific embodiments of the invention may use an asymmetric key scheme, a symmetric key scheme, or some other scheme as may be devised.

In accordance with one embodiment of the invention, a system and methods are provided for generating, storing and validating electronic tickets for multiple venues. The tickets are illustratively stored on a standard smart card, although other devices are also contemplated such as the PalmPilot by 3COM Corporation or the iButton by Dallas Semiconductor. The stored tickets may be for any occasions for which admission or passage may be pre-purchased, such as sporting events, entertainment events, airline flights, automobile tolls, etc. Each venue for which a ticket has been stored on a smart card in accordance with a present embodiment of the invention has an associated applet stored on the smart card. A shared ticketing applet is also stored. These applets are used, as described below, to interface between the smart card and ticket/venue loading facilities and between the smart card and ticket validation devices.

FIG. 1 depicts an illustrative system for issuing, storing and validating tickets stored on a user's smart card in an embodiment of the invention. Smart card **100** may comply with the ISO 7816 specification for smart cards. As such, it is capable of storing various types and amounts of electronic data for later retrieval.

Applet loader **102** loads one or more applets onto smart card **100**. The applets that are loaded onto smart card **100** by applet loader **102** enable smart card **100** to store tickets to venues associated with the loaded applets. For example, one venue applet may correspond to baseball games hosted by the San Francisco Giants. Loading this applet enables smart card **100** to store tickets for specific games or a range of games (e.g., a season pass). Illustratively, applet loader **102** is configured to load an applet pertaining to a single venue. In an alternative embodiment, however, applet loader **102** loads applets from multiple venues.

In addition to venue applets (i.e., applets associated with individual venues), a shared ticketing applet is also loaded onto smart card **100** for use by all venue applets. As discussed below, this shared applet provides functions commonly available to, and used on behalf of, each of the venue applets.

Ticket loader **104** loads electronic tickets for individual events (or a range of events) onto smart card **100**. Each smart card is capable of storing multiple tickets, for the same or different events, venues, dates, etc. Illustratively, each ticket loaded onto smart card **100** is stored in association with the venue applet corresponding to the venue that is hosting the event and will accept the ticket. In a present embodiment, a venue's applet is loaded onto smart card **100** (e.g., by applet loader **102**) before a ticket for an event at that venue is loaded.

Illustratively, ticket validation device **106** is located at a venue hosting an event for which a ticket is stored on smart card **100**. Validation device **106** validates the ticket to ensure that it is for a current event and accepts the ticket based upon this validation.

In a present embodiment of the invention, applet loader **102**, ticket loader **104** and validation device **106** are separate electronic systems equipped to accept, read from, and write to smart card **100**. In this embodiment, a user physically presents smart card **100** to each system in order to effect the desired transaction. In an alternative embodiment, any or all of applet loader **102**, ticket loader **104** and validation device **106** are co-located, particularly the applet loader and ticket loader.

In yet a further alternative embodiment of the invention, any or all of applet loader **102**, ticket loader **104** and validation device **106** comprise a computer system connected to the Internet or other wide area network. In such an embodiment, these systems are accessed by the user through a user computer system that is equipped to accept, read from, and write to smart card **100**.

FIG. 2 depicts smart card **100** populated with the shared ticketing applet, multiple venue applets, and multiple tickets. Smart card **100** incorporates operating system **200** to interface with other devices (such as applet loader **102**, ticket loader **104** and validation device **106** from FIG. 1) and manage the storage and retrieval of information from the smart card. Operating system **200** includes, in the illustrated embodiment, a Java Virtual Machine (JVM) for operating loaded applets. Operating system **200** further includes cryptographic key **200*a*** (hereinafter termed the "applet loader key") for validating applets loaded onto smart card **100**. Thus, applet signatures **202*b*, 210*b*** and **220*b*** are authenticated with applet loader key **200*a*** when the applets are loaded. Illustratively, applet signatures are created prior to, or concurrent with, the loading of the associated applet.

Shared ticketing applet **202** comprises instructions (e.g., in the form of modules, objects, functions, etc.) called upon by the various venue applets installed on smart card **100**. Shared ticketing applet **202** provides functions common to each venue applet (e.g., ticket validation, protocols for communicating with ticket loader **104** and validation device **106**) and therefore allows each venue applet to be smaller in size, thus conserving storage space on smart card **100**. For example, in one embodiment of the invention shared ticketing applet **202** provides instructions for loading a ticket, validating a ticket, and/or canceling a ticket (e.g., after it has been used to gain admittance to an event). Shared ticketing applet **202** includes cryptographic key **202*a*** (hereinafter termed the "venue loader key") to validate individual venue applets, as described below. In particular, when a venue applet is loaded, shared ticketing applet **202** authenticates each applet's venue signature.

In an alternative embodiment of the invention, shared ticketing applet **202** comprises instructions for enforcing or ensuring adherence to ticket details. For example, in such an embodiment smart card **100** could be inserted into a smart card reader located within a seating area at an event to verify that a user is in his or her ticketed seat or to help him or her find the correct seat.

Venue applets **210, 220** are shown installed on smart card **100**. Venue applet **210** illustratively represents home baseball games of the San Francisco Giants. Venue applet **220** illustratively represents United Airlines flights. Venue applets **210, 220** include cryptographic keys **210*a*, 220*a*** (hereinafter termed "venue keys") that are used to authenticate venue applets **210, 220** to ticket loader **104** prior to loading a ticket. Venue keys are also used to validate ticket signatures that accompany tickets for the associated venue.

Venue applets **210, 220** also include applet signatures **210*b*, 220*b*** for validating the venue applets to operating system **200**. As discussed above, applet signatures are illustratively created by applet loader **102** prior to, or concurrent with, the loading of venue applets. Operating system **200** then authenticates applet signatures **210*b*, 220*b*** with applet loader key **200*a*** when the applets are loaded.

Venue applets **210**, **220** further include venue signatures **210**c, **220**c for validating the venue applets to the shared ticketing applet. Similar to applet signatures **210**b, **220**b, venue signatures **210**c, **220** are created prior to or concurrent with the installation of venue applets **210**, **220**. When the venue applets are loaded, shared ticketing applet **202** authenticates the venue signatures.

Tickets **212**, **214**, **216** represent particular home ballgames played at the San Francisco Giants venue. Ticket **222** represents a particular flight offered by United Airlines, from San Francisco to Pittsburgh, Pa.

Each ticket stored on smart card **100** includes information concerning the related event. Thus, tickets **212**, **214** and **216** include information such as the date of the game, opponent and an assigned seat number. The information stored in a ticket is used with the ticket signature, in a present embodiment of the invention, to validate the authenticity of the ticket. Thus, the amount and type of information stored with a ticket varies depending upon the venue, event, type of ticket, etc. Instead of individual tickets **212**, **214** and **216**, the owner of smart card **100** may, for example, have just one ticket in the form of a season pass. The season pass ticket is good for more than one date and will therefore include information different from tickets **212**, **214**, **216**.

Each of tickets **212**, **214**, **216** and **222** includes a ticket signature (represented by the numerals **212**a, **214**a, **216**a and **222**a) generated by ticket loader **104** with a key of the corresponding venue. In an embodiment of the invention using public key encryption (PKE) and asymmetric key pairs, and where venue keys **210**a, **220**a are public venue keys, the ticket signatures are generated using the private keys corresponding to the public keys. In an alternative embodiment using symmetric keys (e.g., DES), ticket loader **104** signs issued tickets with a copy of venue keys **210**a, **220**a. As mentioned above, when a ticket is loaded onto smart card **100**, the corresponding venue applet validates the ticket by authenticating the ticket signature with its venue key.

One skilled in the art will recognize that an applet stored on smart card **100** is able to keep data private and thus inaccessible to other stored applets. This prevents one applet from corrupting or examining tickets associated with a particular venue applet. In a present embodiment, however, tickets are cancelled or deactivated after being presented to validation device **106**. In an alternative embodiment, individual tickets are deleted or overwritten.

Loading an Applet

In a present embodiment of the invention, the venue applets and the shared ticketing applet that are loaded onto smart card **100** comprise executable computer programs or modules of executable computer code. In a present embodiment of the invention, the shared ticketing applet is substantially identical from one smart card to another. Further, each venue's venue applets are similar from one smart card to another, except for the venue key and any tickets that may be loaded.

In one embodiment of the invention, venue applets comprise Java applications constructed according to a standard method. For example, a file containing the Java programming instructions is compiled with a Java compiler to form a binary class file. The class file is then converted into a smart card application file. During this conversion process, the card application file is digitally signed using applet loader key **200**a (shown in FIG. **2**) or its complement, depending upon the type of cryptographic encryption (e.g., symmetric or asymmetric).

FIG. **3** depicts an illustrative process by which a signed card application file (e.g., applet **210** in FIG. **2**) is loaded onto smart card **100** from applet loader **102**. Applet loader **102** is, in a present embodiment of the invention, a ticket vending machine and is co-located with ticket loader **104**. In this embodiment, venue applet **210** is automatically loaded when a Giants' baseball ticket is purchased, unless the applet is already resident on smart card **100**. Also, in this embodiment shared ticketing applet **202** is automatically loaded if not resident on smart card **100**. In an alternative embodiment, either or both of shared ticketing applet **202** and venue applet **210** are pre-loaded on smart card **100** at the time it is manufactured or the time it is sold.

With reference now to FIG. **3**, state **300** is a start state. In state **302**, applet loader **102** is coupled to smart card **100** and prepares to download applet **210**. Illustratively, the owner of smart card **100** inserts the smart card into a device comprising applet loader **102** and selects applet **210** for installation (e.g., by indicating a desire to purchase Giants baseball tickets). In an alternative embodiment, the owner inserts smart card **100** into a separate computer system connected to applet loader **102** via the Internet or other communication link.

In state **304**, smart card **100** indicates that it is prepared to load an applet and, in a present embodiment, passes the applet loader information concerning its present configuration (e.g., which applets are loaded, which versions of operating system and Java Virtual Machine are installed). In one embodiment, smart card **100** performs a self-check prior to indicating that it is ready to receive an applet. Illustratively, the self-check tests the card's ability to store and retrieve data and tests for bad or damaged memory cells. Information transmitted to applet loader **102** by the smart card may include the amount of storage space available on the card. If insufficient space exists for loading the selected applet, an error message is displayed for the user.

In state **306**, applet loader **102** determines whether shared ticketing applet **202** is already resident on smart card **100**. As described above, shared ticketing applet **202** contains instructions used by venue applet **210** and other venue applets. Illustratively, this determination is made based upon information returned to applet loader **102** by smart card **100** in state **304**.

If it is determined in state **306** that shared ticketing applet **202** is not installed on smart card **100**, the process continues with state **310**. Otherwise, in state **308** it is determined whether venue applet **210** is already loaded on smart card **100**. If not, the process proceeds to state **316**. If, however, both applets are already loaded, the process exits in end state **320**.

In state **310** the shared ticketing applet is signed (e.g., by applet loader **102**), if not already signed, with a cryptographic key complementary to applet loader key **200**a (e.g., when using an asymmetric encryption scheme, a "private" key corresponding to "public" key **200**a) to create applet signature **202**b (shown in FIG. **2**). The signed applet is then downloaded to smart card **100**. Illustratively, applets are downloaded and stored on the smart card in multiple streams of bytes (e.g., approximately 200 bytes in each stream), and each stream is validated by an associated checksum. In state **312**, the smart card validates accurate receipt of the applet and, in state **314**, informs the applet loader whether the installation was successful or not. If shared applet **202** could not be correctly loaded, an error message is returned and the process ends at end state **320**.

If the installation of shared ticketing applet **202** was successful or, if it was determined in state **308** that venue applet **210** has not been loaded, the process continues at state **316**.

In state **316**, venue applet **210** is signed (if not already signed) by applet loader **102** to create applet signature **210*b*** and/or venue signature **210*c*** and is then downloaded onto smart card **100** from applet loader **102**. Venue key **210*a***, as discussed below, will be used to authenticate venue applet **210** to ticket loader **104** and to validate tickets loaded from the ticket loader. Depending upon the type of cryptographic security that is preferred (e.g., symmetric or asymmetric keys), applet signature **210*b*** and venue signature **210*c*** are created with applet loader key **200*a*** and venue loader **202*a***, respectively, or with their complements.

In state **318**, smart card **100** validates the downloaded applet and indicates to the applet loader that it was successfully loaded or that an error was encountered. Illustratively, smart card **100** validates successful receipt of the applet by computing a checksum and comparing it to a checksum provided by applet loader **102**. In an alternative embodiment, applet signature **210*b*** of the downloaded applet is validated using a cryptographic technique corresponding to the form of the key used to create the signature. In one particular such embodiment, smart card **100** computes a hash value from the applet and compares it to a hash value retrieved from the signature. If they match, the smart card considers the applet to have been received intact. A similar process is used to validate a ticket signature when a ticket is downloaded. The process then ends at end state **320**.

Loading a Ticket

Once a venue applet is loaded onto smart card **100**, tickets for events at that venue (e.g., matches or games at a sporting field, flights offered by an airline) may be purchased and loaded as well. Venue applets, shared ticketing applet **202** and related tickets are, in a present embodiment of the invention, loaded in conjunction with each other, as necessary, from a combined ticket/applet loader.

FIG. **4** depicts an illustrative procedure for purchasing an electronic ticket to a Giants baseball game (for which venue applet **210** has been installed) from ticket loader **104** and installing it on smart card **100**. In a present embodiment of the invention, ticket loader **104** is part of a web server connected to a public network such as the Internet. In this embodiment, smart card **100** is coupled to a computer system operated by the owner of smart card **100** that is also connected to the Internet. Tickets are selected using an interface for the venue's web server, and then downloaded over the Internet and stored on smart card **100**.

With reference now to FIG. **4**, state **400** is a start state. In state **402**, the owner of smart card **100** initiates the ticket purchasing/loading procedure. In one embodiment of the invention, the owner first selects an event for which a ticket is desired. In the presently described embodiment, for example, a baseball game is identified along with the number and type of seats desired. As another example, the owner identifies to an airline reservation agent a flight the owner wishes to take, including a date and time and perhaps a seat. After the smart card owner selects his or her venue/event and specifies any necessary or criteria concerning the event, he or she signals acceptance of the ticket as configured.

In state **404**, ticket loader **104** identifies itself to and challenges smart card **100** in order to authenticate the card and/or venue applet **210**. Illustratively, the challenge is a "zero knowledge proof" taking the form of a random number transmitted to smart card **100** by ticket loader **104**. In state **406**, venue applet **210** meets the challenge by generating a digital signature with venue key **210*a***, and returning the result to ticket loader **104**. In an alternative embodiment, venue applet **210** meets the challenge in step **406** by encrypting the random number with venue key **210*a*** and returning the result to ticket loader **104**.

In state **408**, ticket loader **104** validates the signature received from smart card **100**. For purposes of this validation, ticket loader **104** possesses a key complementary to venue key **210*a***. For example, in an embodiment of the invention employing asymmetric keys (e.g., RSA), wherein venue key **210*a*** is a public key of the associated venue, ticket loader **104** possesses the corresponding private key. In an embodiment of the invention using symmetric keys (e.g., Digital Encryption Standard), ticket loader **104** and venue applet **210** possess copies of the same key. If the validation attempt fails, the ticket loading process either attempts the challenge/validation procedure again (up to a limited number of times) or fails and reports an error, depending upon the implementation and security concerns.

Next, in state **410** ticket loader **104** generates and signs ticket **212** for the venue based upon the event data selected by the smart card owner/user. Illustratively, ticket loader **104** signs ticket **212** using the same key with which venue applet **210** was validated in state **408**. In state **412**, ticket **212**, complete with signature **212*a***, is downloaded and stored on smart card **100**.

In state **414**, venue applet **210** validates downloaded ticket **212** by authenticating signature **212*a*** with venue key **210*a*** and respond with a message indicating success or failure. In an alternative embodiment of the invention, a second venue key, different from venue key **210*a*** is stored with venue applet **210** for the purpose of validating downloaded tickets. The procedure ends with end state **416**.

In the presently described embodiment, the process described above must be followed for each ticket downloaded from ticket loader **104**. In an alternative embodiment, multiple tickets may be selected, processed and downloaded for a single venue at a time.

Validating a Ticket

In a present embodiment of the invention, tickets are validated by validation device **106** when presented for acceptance at the appropriate venue. FIG. **5** depicts an illustrative procedure for validating ticket **212** in accordance with a present embodiment of the invention.

State **500** is a start state. In state **502**, a user presents smart card **100** to validation device **106** in order to gain admittance to the ball game identified in ticket **212**. Illustratively, validation device **106** comprises a computer system configured to accept and communicate with smart card **100**.

In state **504**, validation device **106** generates and issues a challenge to smart card **100** as was done in the ticket loading procedure described above. The random number provided to smart card **100** is signed by venue applet **210**, using venue key **210*a***, in state **506**. In state **508**, the validation device authenticates the signature using a key complementary to venue key **210*a***. By authenticating the signature returned with the challenge, validation device **106** is able to validate venue applet **210**.

After authenticating the signature, in state **510** validation device **106** requests ticket data retained by smart card **100**. Venue applet **210** transmits ticket **212** (e.g., the ticket data and signature) to validation device **106** in state **512**. Illustratively, validation device **106** is only informed of stored ticket(s) usable for a current event, which are identified by the date, time and/or other identifying data. In one embodiment of the invention, shared ticketing applet **202** determines the ticket(s) to be identified to validation device **106** (e.g., by determining which venue—and therefore which venue applet and tickets—corresponds to the validation device). Alternatively, venue applet **210** and validation device **106** communicate in order to determine which, of multiple tickets associated with the present venue, should be used.

In state **514**, validation device **106** verifies the ticket data (e.g., confirms the date, time, participating teams, seat number) and authenticates the ticket signature. If the ticket data and signature pass inspection, smart card **100** is instructed to cancel or erase ticket **212** and the user is admitted.

In the presently described embodiment of the invention, ticket **212** will be overwritten with a future ticket loaded onto smart card **100**. In an alternative embodiment, tickets are not erased or overwritten.

The foregoing descriptions of embodiments of the invention have been presented for purposes of illustration and description only. They are not intended to be exhaustive or to limit the invention to the forms disclosed. Many modifications and variations will be apparent to practitioners skilled in the art. Accordingly, the above disclosure is not intended to limit the invention; the scope of the invention is defined by the appended claims.

What is claimed is:

1. A method of using an electronic device to store tickets, comprising:

receiving a first venue module associated with a first venue, said first venue module including a first venue key for validating a ticket for said first venue;

validating said first venue module with a module loader key of the electronic device;

receiving a shared module said shared module comprising an instruction used by said first venue module;

validating said shared module with said module loader key;

receiving a first ticket for said first venue;

receiving a first ticket signature associated with said first ticket; and

authenticating said first ticket signature with said first venue key.

2. The method of claim **1**, further comprising:

receiving a second venue module associated with a second venue, said second venue module including a second venue key for validating a ticket for said second venue;

validating said second venue module with said module loader key;

receiving a second ticket for an event offered at said second venue;

receiving a second ticket signature with said second ticket; and

authenticating said second ticket signature with said second venue key;

wherein said first venue is different from said second venue.

3. The method of claim **1**, wherein each of said first venue module and said shared module include a module signature and wherein said validating comprises authenticating the module signature of the validated module with the module loader key.

4. The method of claim **1**, wherein said shared module includes a shared venue key for validating a venue module, further comprising validating said first venue module with said shared venue key.

5. The method of claim **1**, wherein said receiving a first ticket comprises:

receiving a challenge from a ticket loader;

signing said challenge with said first venue key; and

transmitting said signed challenge to said ticket loader.

6. The method of claim **1**, wherein said receiving a first venue module comprises:

receiving a first series of instructions for processing a ticket for an event at a first venue;

receiving a first venue key for said first venue;

storing said series of instructions; and

storing said first venue key in association with said series of instructions.

7. The method of claim **6**, further comprising:

determining whether said shared module is stored on the electronic device; and

replacing said shared module if said shared module is stored on the electronic device.

8. The method of claim **6**, wherein said receiving a shared module comprises:

receiving a second series of instructions used by one or more venue modules;

receiving a venue loader key for validating said one or more venue modules;

storing said second series of instructions; and

storing said venue loader key in association with said second series of instructions.

9. The method of claim **1**, wherein said validating said first venue module comprises authenticating a module signature of said first venue module with a module loader key of the electronic device.

10. The method of claim **1**, further comprising canceling said first ticket.

11. The method of claim **10**, wherein said canceling said first ticket comprises marking said first ticket invalid.

12. The method of claim **7**, wherein said replacing said shared module comprises:

marking said shared module invalid; and

receiving a new version of said shared module.

13. The method of claim **1**, further comprising providing said first ticket to a validation device of said first venue.

14. A method of maintaining tickets for multiple venues on an electronic device, comprising:

storing a first venue module, wherein said first venue module is associated with a first venue and includes a first venue key;

storing a shared module, said shared module comprising an instruction used by said first venue module and having a shared venue key for validating said first venue module;

validating said shared module;

receiving a challenge from a ticket loader;

signing said challenge with a first digital signature using said first venue key;

transmitting said signed challenge to said ticket loader;

receiving a first electronic ticket for said first venue;

receiving a first ticket signature, said first ticket signature being associated with said first electronic ticket; and

authenticating said first ticket signature with said first venue key.

15. The method of claim **14**, further comprising:

storing a second venue module, wherein said second venue module is associated with a second venue and includes a second venue key;

wherein said second venue is different from said first venue.

16. The method of claim **15**, further comprising:

receiving a second electronic ticket for said second venue;

receiving a second ticket signature, said second ticket signature being associated with said second electronic ticket; and

authenticating said second ticket signature with said second venue key.

17. The method of claim 14, wherein said receiving a challenge comprises receiving a randomly generated number.

18. The method of claim 14, wherein said receiving a first electronic ticket comprises receiving one or more details of an event at said first venue.

19. The method of claim 14, further comprising:

receiving a second challenge from a validation device at said first venue;

signing said second challenge using said first venue key;

transmitting said signed second challenge to said validation device; and

transmitting said first electronic ticket.

20. The method of claim 19, further comprising canceling said first electronic ticket.

21. The method of claim 19, wherein said receiving a second challenge comprises receiving a randomly generated number.

22. The method of claim 19, wherein said transmitting said first ticket comprises transmitting one or more details comprising said first electronic ticket.

23. The method of claim 14, wherein said validating comprises validating said first venue module with one or more of said shared venue key and a module loader key stored on the electronic device.

24. An electronic device for storing tickets, said device comprising a memory configured to store:

a shared module, said shared module comprising an instruction used by one or more venue modules and having a shared venue key configured for validating said one or more venue modules;

a first venue module associated with a first venue, said first venue module including a first venue key for validating tickets for said first venue;

a module loader key for validating one or more of said shared module and said first venue module; and

a first ticket for said first venue, said first ticket having a first ticket signature, wherein said first ticket signature is authenticatable with said first venue key.

25. A computer readable storage medium storing instructions that, when executed by a computer, cause the computer to perform a method of using an electronic device to store a ticket, the method comprising:

receiving a first venue module associated with a first venue, said first venue module including a first venue key for validating a ticket for said first venue;

validating said first venue module with a module loader key of the electronic device;

receiving a shared module, said shared module comprising an instruction used by said first venue module;

validating said shared module with said module loader key;

receiving a first ticket for said first venue;

receiving a first ticket signature associated with said first ticket; and

authenticating said first ticket signature with said first venue key.

* * * * *

US006390374B1

(12) United States Patent
Carper et al.

(10) Patent No.: US 6,390,374 B1
(45) Date of Patent: May 21, 2002

(54) SYSTEM AND METHOD FOR INSTALLING/DE-INSTALLING AN APPLICATION ON A SMART CARD

(76) Inventors: Todd Carper, 19672 Stevens Creek Blvd., Cupertino, CA (US) 95014; David Hemmo, 12 Rue Caillaux, 75013 Paris (FR); An Van Le, 7056 Anjou Creek Cir., San Jose, CA (US) 95120

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: 09/386,288

(22) Filed: Aug. 31, 1999

Related U.S. Application Data

(60) Provisional application No. 60/116,243, filed on Jan. 15, 1999.

(51) Int. Cl.[7] G06K 19/06

(52) U.S. Cl. 235/492; 235/380

(58) Field of Search 235/492, 380, 235/384, 441, 375, 487, 486; 360/23, 29; 705/40–44

(56) References Cited

U.S. PATENT DOCUMENTS

4,727,244 A * 2/1988 Nakano et al. 235/380
4,882,474 A 11/1989 Anderl et al. 395/182
5,070,233 A * 12/1991 Takizawa et al. 238/380
5,310,999 A * 5/1994 Claus et al. 235/384
5,740,349 A 4/1998 Hasbun et al. 235/380
5,805,869 A * 9/1998 Smith et al. 395/507
5,901,303 A * 5/1999 Chew 395/400

OTHER PUBLICATIONS

Rankl et al., Smart Card Handbook, John Wiley & Sons, (copyright 1997), pps, 14, 91, 103, 107–10, and 127–28.

Guthery et al., Smart Card Developer's Kit, Macmillan Technical Publishing (copyright 1998), pps, 175, 176, and 219–20.

PCT International Search Report Apr. 2000.

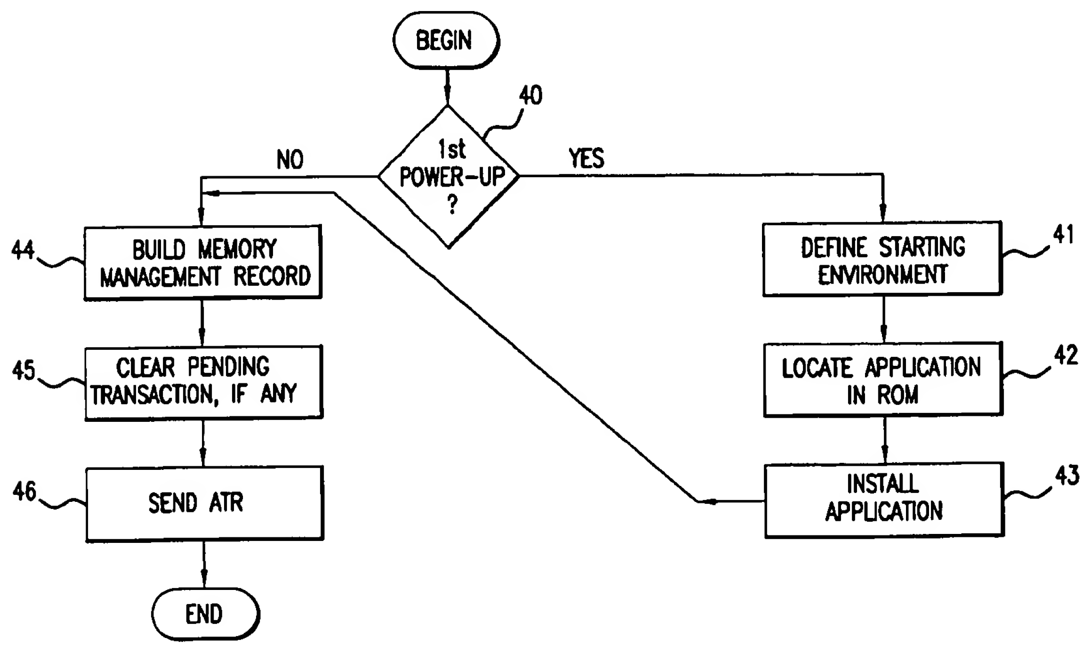
* cited by examiner

Primary Examiner—Thien M. Le

(57) ABSTRACT

A set of related routines allows an application to be securely installed on, or de-installed from a smart card or other portable token. The capabilities of a true operating system on the smart card are used to facilitate installation/de-installation.

31 Claims, 9 Drawing Sheets



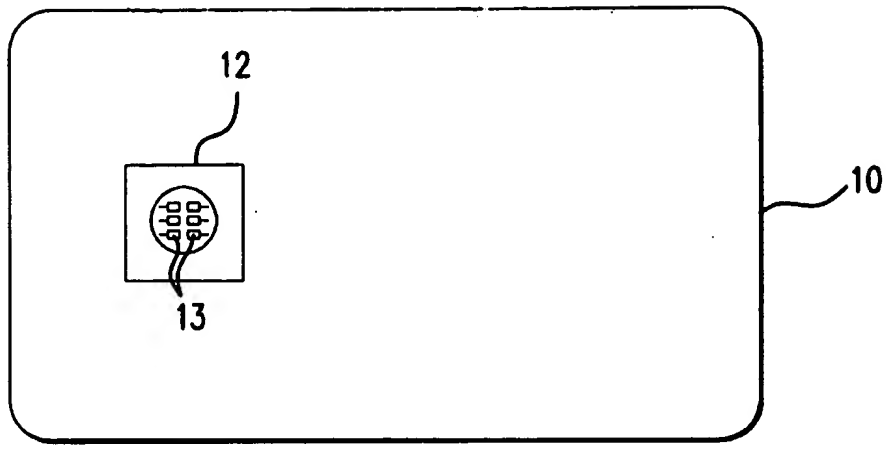

12
10
13

FIG.1

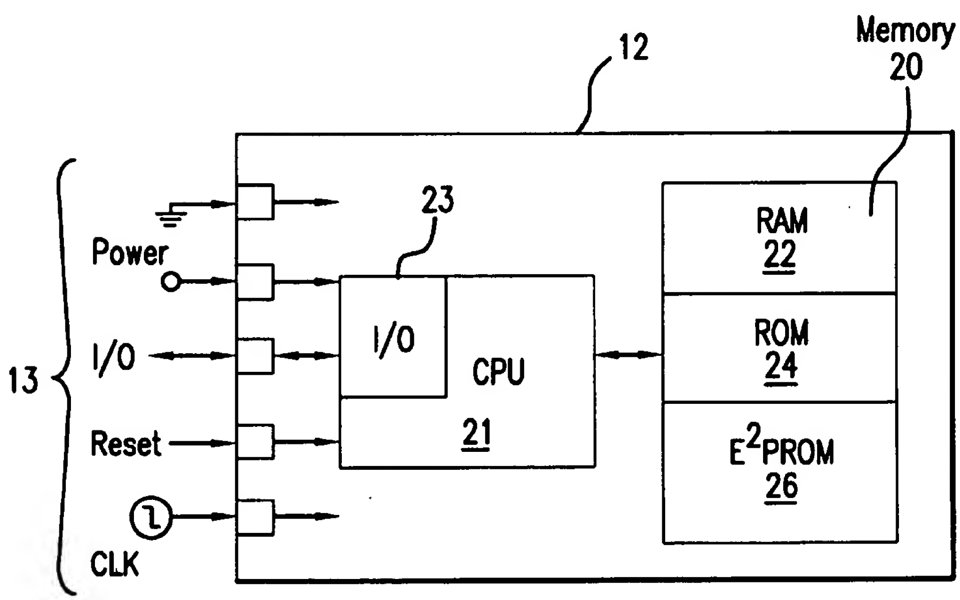

12
Memory
20
23
RAM
22
Power
ROM
24
I/O
I/O
CPU
13
21
Reset
E2PROM
26
CLK

FIG.2

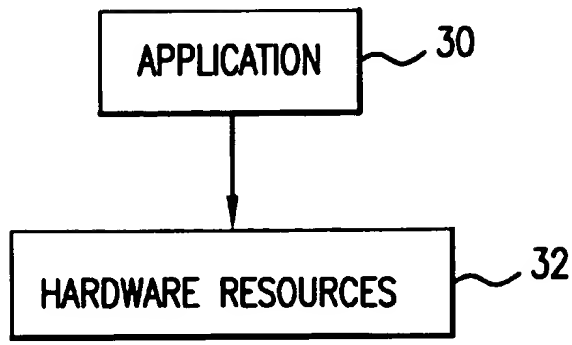

APPLICATION
30
HARDWARE RESOURCES
32

FIG.3A

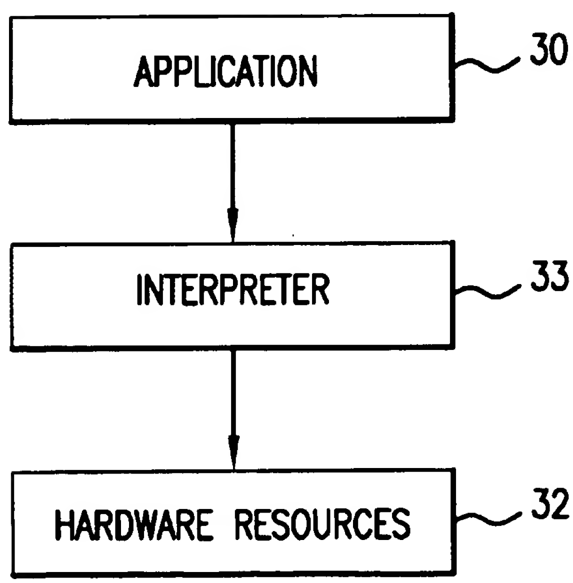

APPLICATION
30
INTERPRETER
33
HARDWARE RESOURCES
32

FIG.3B

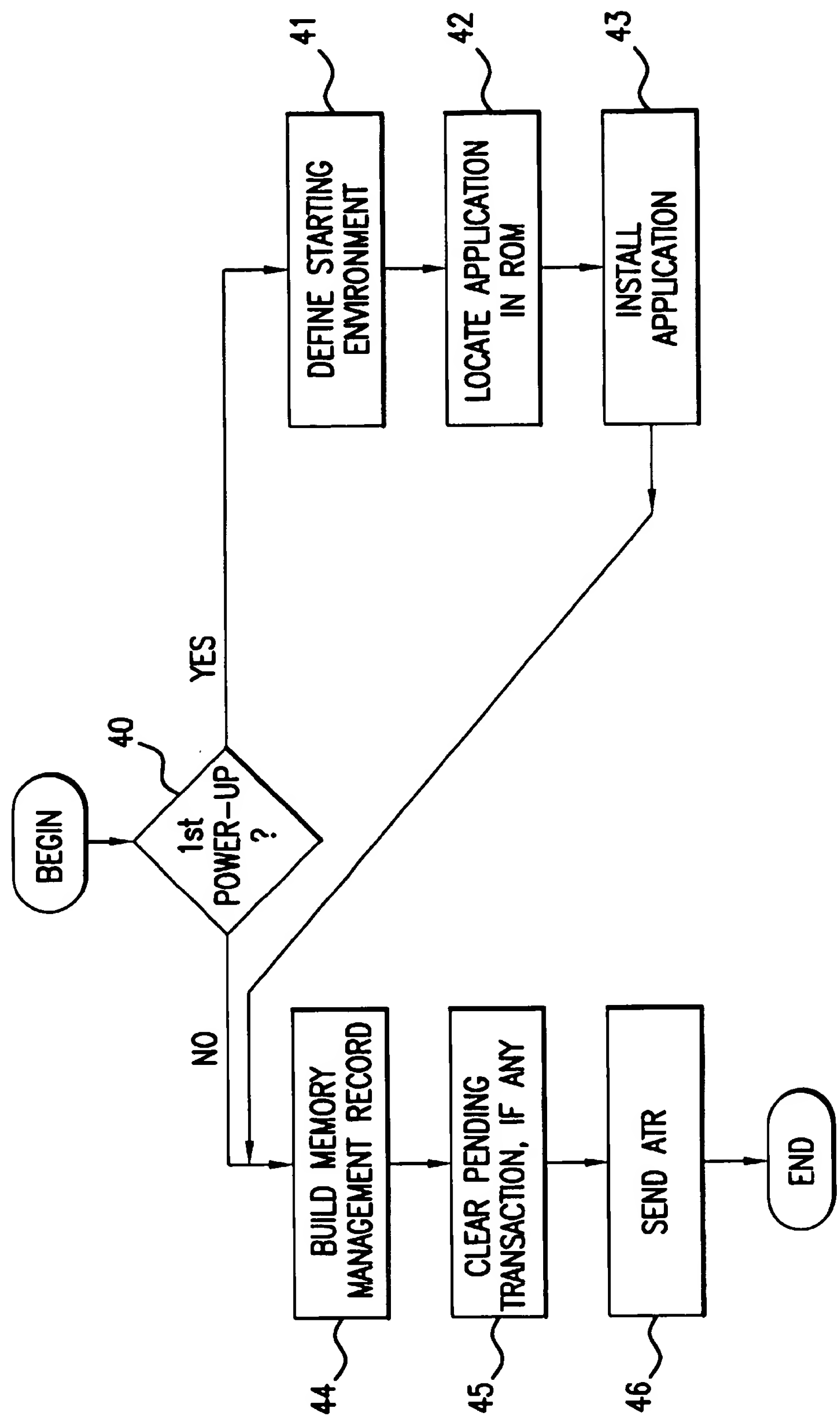

BEGIN
1st POWER-UP ?
40
YES
NO
DEFINE STARTING ENVIRONMENT
41
LOCATE APPLICATION IN ROM
42
INSTALL APPLICATION
43
BUILD MEMORY MANAGEMENT RECORD
44
CLEAR PENDING TRANSACTION, IF ANY
45
SEND ATR
46
END

FIG.4

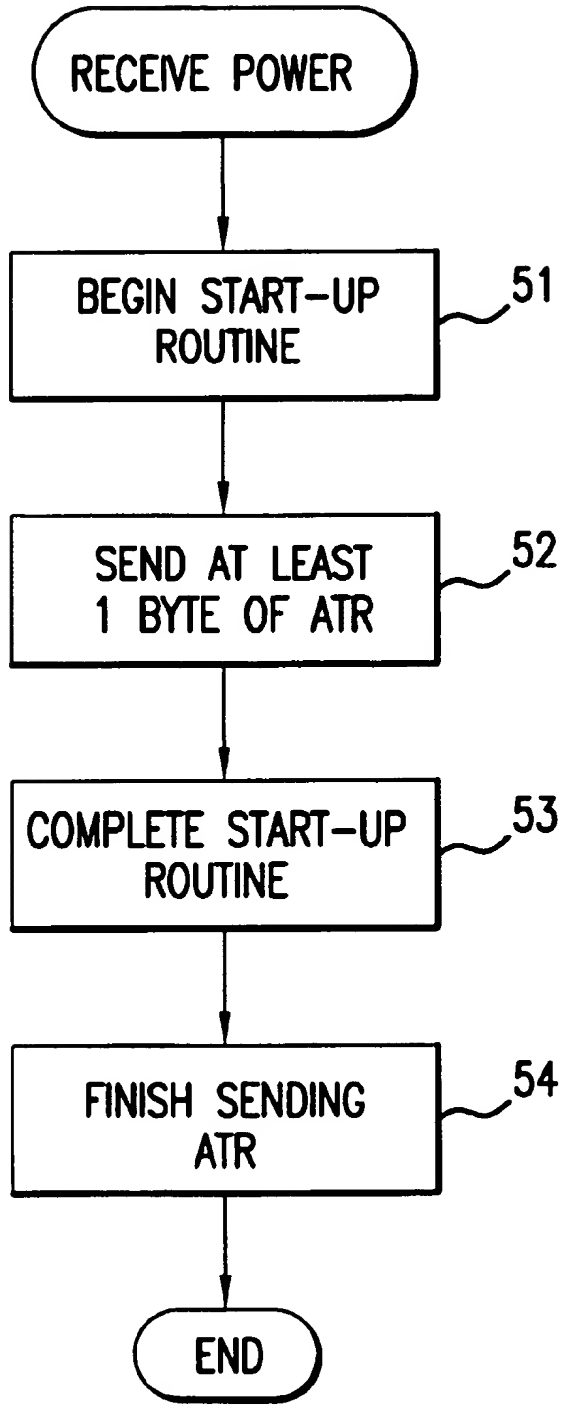

RECEIVE POWER
BEGIN START-UP ROUTINE
51
SEND AT LEAST 1 BYTE OF ATR
52
COMPLETE START-UP ROUTINE
53
FINISH SENDING ATR
54
END

FIG.5

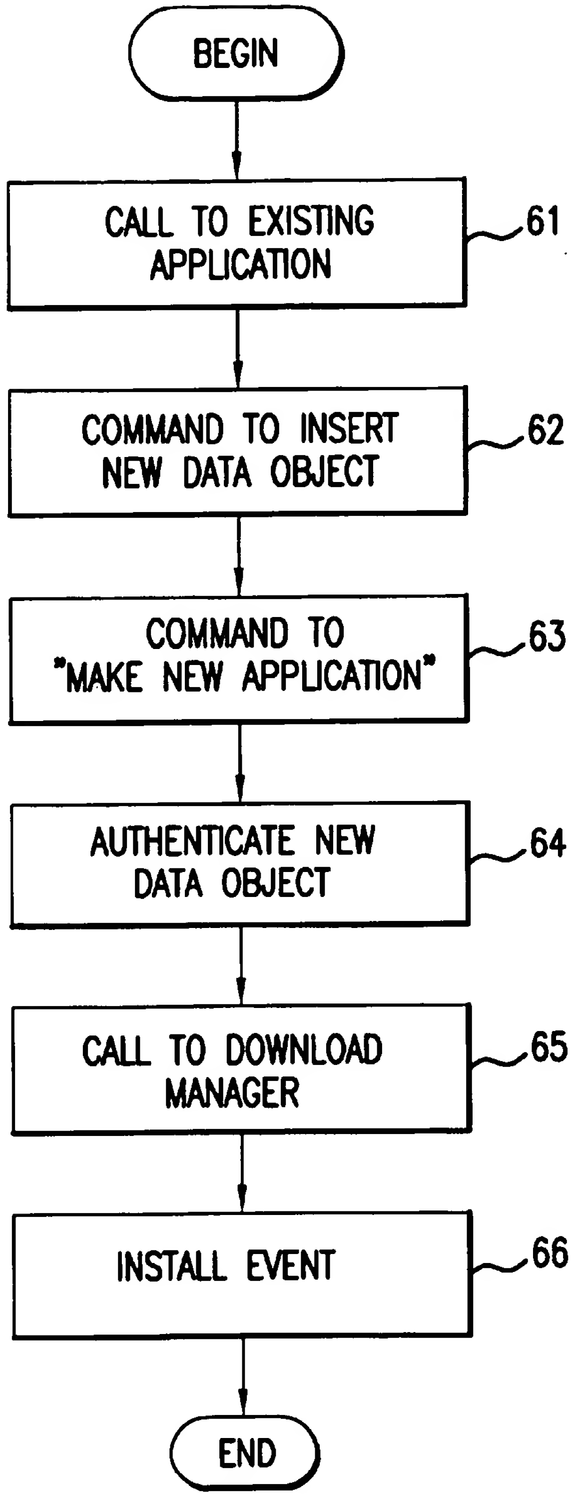

BEGIN
CALL TO EXISTING APPLICATION
61
COMMAND TO INSERT NEW DATA OBJECT
62
COMMAND TO "MAKE NEW APPLICATION"
63
AUTHENTICATE NEW DATA OBJECT
64
CALL TO DOWNLOAD MANAGER
65
INSTALL EVENT
66
END

FIG.6

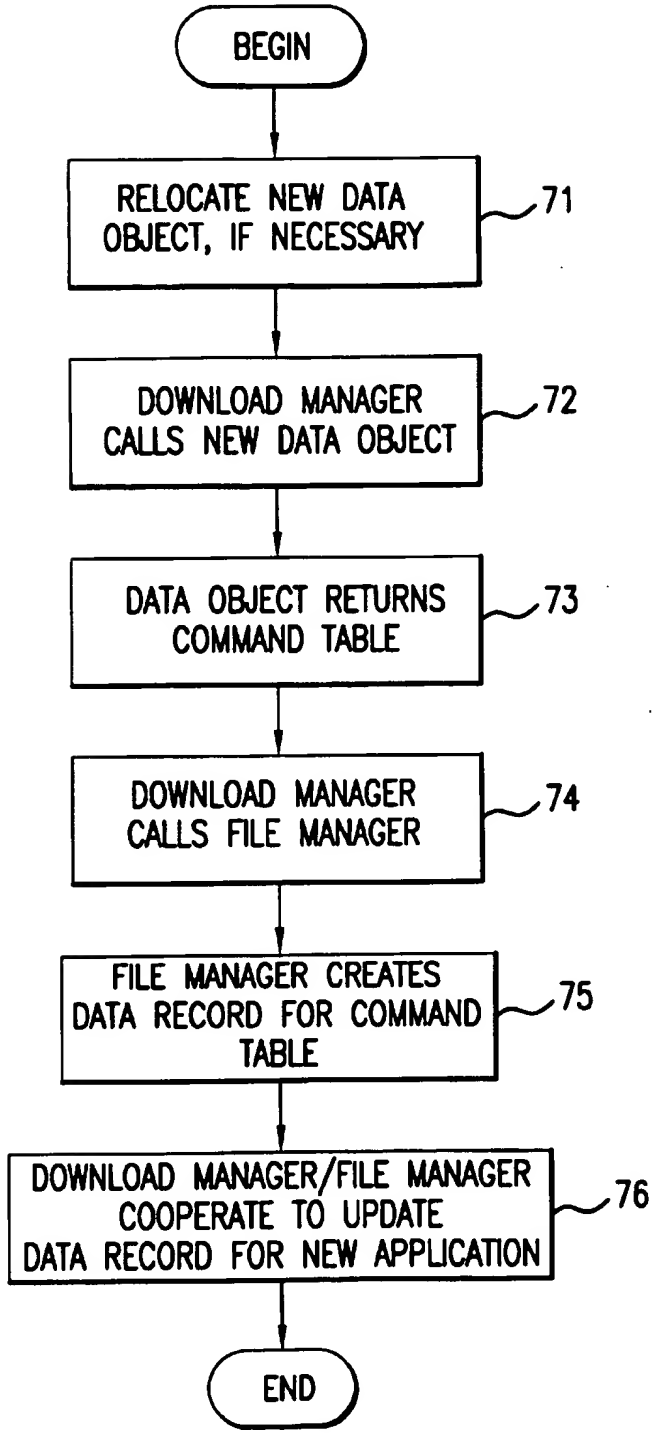

BEGIN
RELOCATE NEW DATA OBJECT, IF NECESSARY
71
DOWNLOAD MANAGER CALLS NEW DATA OBJECT
72
DATA OBJECT RETURNS COMMAND TABLE
73
DOWNLOAD MANAGER CALLS FILE MANAGER
74
FILE MANAGER CREATES DATA RECORD FOR COMMAND TABLE
75
DOWNLOAD MANAGER/FILE MANAGER COOPERATE TO UPDATE DATA RECORD FOR NEW APPLICATION
76
END

FIG.7

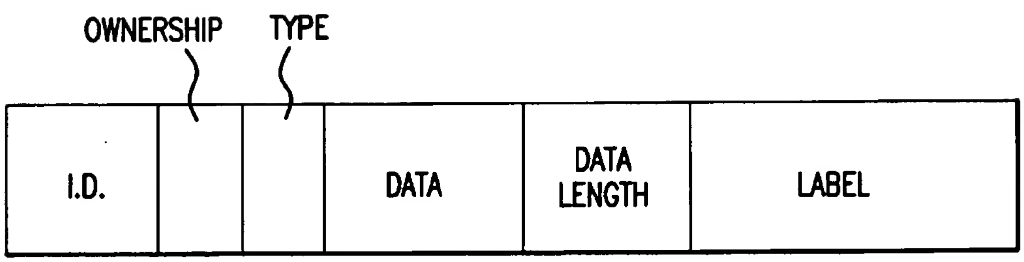


FIG.8

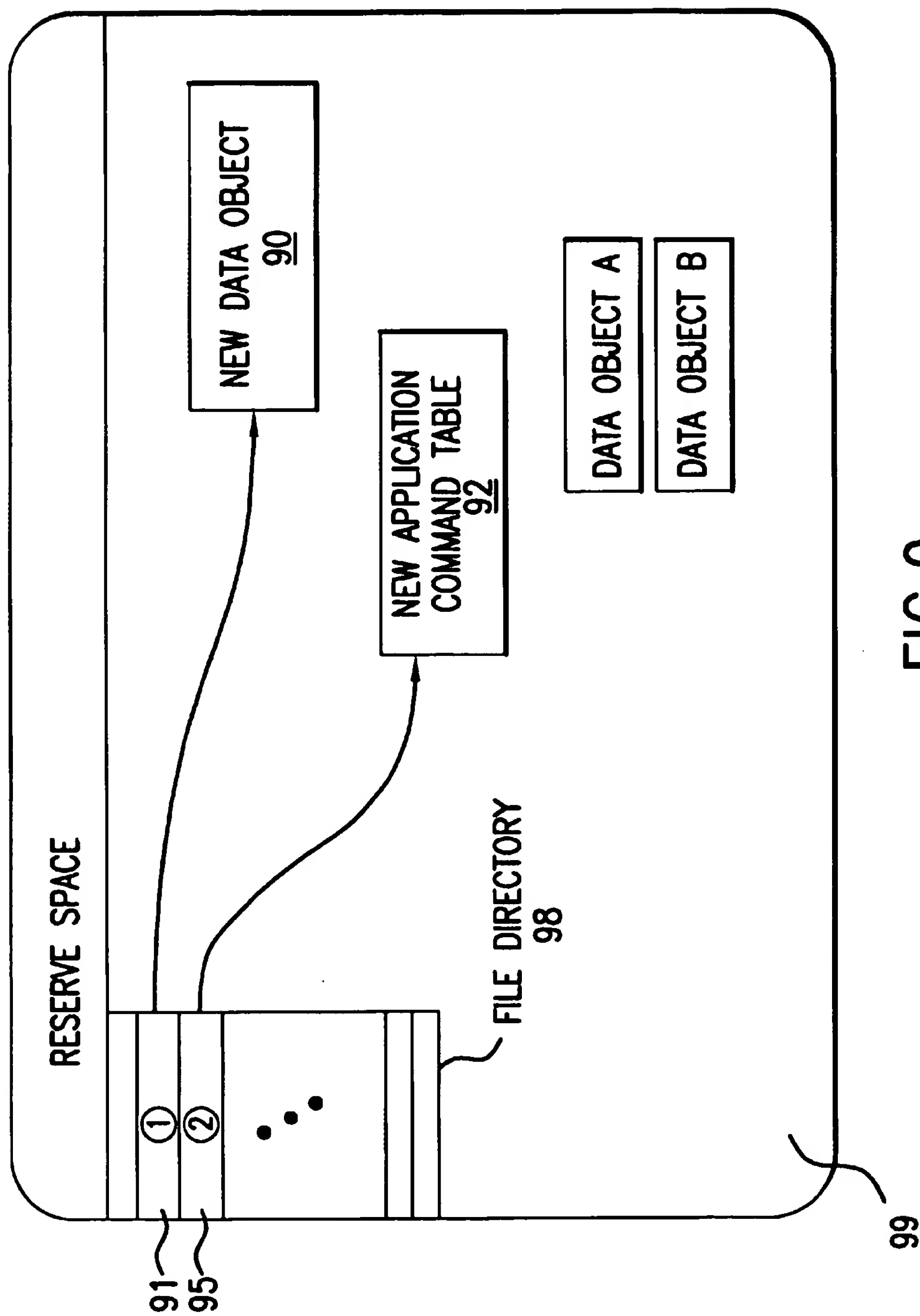

RESERVE SPACE
NEW DATA OBJECT
90
NEW APPLICATION
COMMAND TABLE
92
DATA OBJECT A
DATA OBJECT B
FILE DIRECTORY
98
1
2
91
95
99

FIG.9

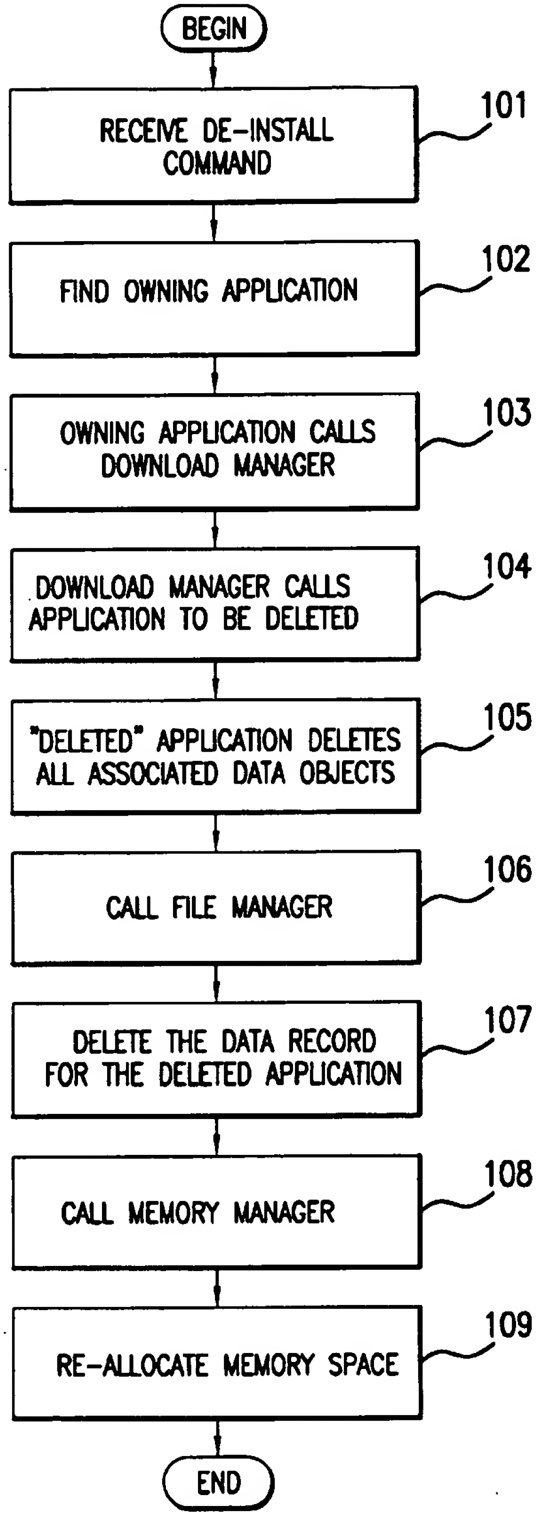

BEGIN
RECEIVE DE-INSTALL COMMAND
101
FIND OWNING APPLICATION
102
OWNING APPLICATION CALLS DOWNLOAD MANAGER
103
DOWNLOAD MANAGER CALLS APPLICATION TO BE DELETED
104
"DELETED" APPLICATION DELETES ALL ASSOCIATED DATA OBJECTS
105
CALL FILE MANAGER
106
DELETE THE DATA RECORD FOR THE DELETED APPLICATION
107
CALL MEMORY MANAGER
108
RE-ALLOCATE MEMORY SPACE
109
END

FIG.10

SYSTEM AND METHOD FOR INSTALLING/DE-INSTALLING AN APPLICATION ON A SMART CARD

CROSS REFERENCE TO RELATED APPLICATION(S)

This application claims the benefit of U.S. Provisional Application No. 60/116,243 filed Jan. 15, 1999.

FIELD OF THE INVENTION

The present invention relates to the field of portable tokens such as smart cards. More particularly, the present invention relates to a smart card capable of dynamically installing or de-installing one or more applications. Installation/de-installation may be done in the field through a terminal, or through an un-secure source such as the Internet.

BACKGROUND OF THE INVENTION

Most consumers are familiar with credit cards, debit cards, and automatic teller machine (ATM) cards. Such cards are increasingly used to access, transfer and spend money. The back of these cards includes a magnetic strip storing encoded information about the cardholder and the account(s) accessible by the card. Terminals, including ATMs and merchant point-of-sale terminals, read the encoded information from the card and access the cardholder's account to complete a transaction.

Besides the well-known credit and debit cards, stored value cards are becoming increasingly popular. Stored value cards are purchased or issued with a specific monetary value. When the cardholder desires to use the stored value card to purchase goods or services, the card is presented at the point of sale and the cost of the goods or services is deducted from the value of the card. The cardholder may continue to use the stored value card in this manner until all the value has been removed from the card. The card may then be discarded, or its value may be replenished. Such cards are commonly used to pay subway fares or to make long distance phone calls.

For many types of transactions, however, the current trend is away from credit/debit cards and stored value cards, and into a class of devices generally called smart cards. Rather than employing information encoded on a magnetic strip, smart cards include a microprocessor and a memory element embedded within a credit card size device. With a microprocessor, a smart card is able to interact with a greater variety of terminals across a broader range of transactions. In this broader range of transactions, the smart card is able to communicate more information regarding the cardholder, cardholder account, transaction authorization, etc.

The term "smart card" is used throughout as a convenient name for a broad class of devices sometimes referred to as portable tokens. Smart cards are the most common present form of portable tokens, but as will be seen hereafter the actual physical form of the portable token, as well as the specific means by which the portable token communicates data to the outside world are not the subject of the present invention.

Smart cards have been used in various applications for some time. FIG. 1 shows an exemplary smart card 10. Roughly the size of a credit card, smart card 10 includes a microprocessor 12 with an integral memory element, and conductive contacts 13. Microprocessor 12 is typically a single wafer integrated circuit mounted on, or embedded within the otherwise plastic smart card. Conductive contacts 13 interface with a terminal to electrically transfer data between the terminal and the smart card. Other embodiments of the smart card do not include conductive contacts 13. Such "contactless" smart cards receive information via proximately coupling, such as magnetic coupling, or via remote coupling, such as radio communication.

The microprocessor 12 and conductive contacts 13 of FIG. 1, are shown in some additional detail in FIG. 2. Conductive contacts variously include power contacts, at least one input/output (I/O) port, a reset port, and a clock (clk) signal port. Microprocessor 12 comprises a central processing unit (CPU) 21 which is generically control logic including I/O circuitry 23. Terminal signals variously interface with CPU 21 through the conductive contacts 13 and I/O circuitry 23. Microprocessor 12 is associated with a memory element 20. The "memory" may be formed on the same integrated circuit as the microprocessor, or may be formed on a separate device. Generally, the memory includes Random Access Memory (RAM) 22, Read Only Memory (ROM) 24, and read/write memory, such as an Electrically Erasable Programable Read Only Memory (EEPROM) 26. However, some or all of these presently-used memory elements may be replaced by battery backed-up RAM, flash memory, or other electronic data storage media.

Operating power, a user input keypad, and a display for the smart card microprocessor are typically provided by a terminal. The term "terminal" broadly indicates any device exchanging information with a smart card using any type or number of data transfer means. A computer, ATM, merchant point-of-sale device, telephone, or security control device, are present examples of terminals.

A broad class of terminals nominally include a mechanism detecting the presence of a properly positioned smart card. Upon detecting the smart card, the terminal provides power to the microprocessor, and typically sends a reset (RST) signal to the smart card. The smart card uses the RST signal to reset itself, or to initiate an internal reset function. After reset, the smart card returns an answer-to-reset (ATR) signal to the terminal. The nature and protocol for the ATR signal is established by International Standards Organization (ISO) standard 7816. As established, the ATR is a multi-byte signal communicating basic information concerning the smart card to the terminal. Once such basic information is successfully recognized by the terminal, communication, i.e., data transfer, between the smart card and the terminal can be established.

Smart cards can be programmed to operate as stored value cards, credit cards, debit cards, ATM cards, calling cards, personal identity cards, critical record storage devices, etc. In these varied capacities, a smart card may, at least in theory, be designed to use a number of different application programs. In actual practice, however, an inability to readily develop and operate applications from a variety of sources has limited the type and number of applications placed on the conventional smart card. In fact, most conventional smart cards include only a single application, or at most a single type of application.

This is not surprising when one considers that from programming and implementation perspectives, a conventional first generation smart cards is little more than an embedded application. Looking at FIG. 3A, such first generation cards can be viewed as an application 30 stored in memory which runs a set of microprocessor-specific instructions on hardware resources 32. The term "hardware resources" is used to generically indicate the memory and

logic circuits, with their associated interfaces, used to execute microprocessor instructions but may also include I/O circuits, power circuits, and the other hardware. Given the structure shown in FIG. 3A, each application must be written in a very low level, or machine level language. This language is specific to the microprocessor on which the application is intended to run.

The first generation, embedded application programming model offers at least one significant advantage—programming flexibility. Microprocessors are typically able to execute a significant set of instructions. Since an embedded application is written at the machine level, the full range of the microprocessor's instructions set may be accessed and utilized by the application.

Unfortunately, such programming flexibility comes at a high price. In order to run an existing application on a different microprocessor, it must often be completely rewritten. Debugging, updating, and testing of embedded applications are arduous. Further, machine level programming is difficult and requires a great deal of hardware specific expertise. Embedded programmers are, thus, hard to find and expensive to retain. All of these factors combine to restrict the number and quality of smart card applications. Further, the hardware specific nature of the resulting applications contributes to the incompatibility problems which characterize conventional smart card applications.

Such conventional smart cards do not employ a true operating system. Rather, a specific application written according to the microprocessor instruction set is stored in ROM and executed in accordance with commands received from a terminal. MPCOS, VisaCash, GSM, and Proton are examples of such first generation embedded applications.

Because conventional first generation smart cards store their embedded application in ROM, there is no real opportunity to significantly change or modify the application once smart cards are fielded. This presents a real problem to smart card issuers, since operating/programming errors and oversights, collectively and generically called "bugs," are continually identified following release of any smart card application. Historically, severe bugs may only be remedied by physically replacing smart cards in the field with cards having an improved version of the application. Card issuers and users generally learn to live with less severe bugs.

Attempts have been made to avoid these unpleasant alternatives by utilizing the EEPROM portion of smart card memory. U.S. Pat. No. 5,542,081 allows "filter instructions" to be placed in the ROM based application which index an address in EEPROM. The EEPROM address thus allows subsequent programming steps to be grafted into the existing application stored in ROM. This additional filtering approach does create a mechanism of sorts for mitigating the effects of application bugs, but it doe not actually correct the errant application code generating the bug.

For example, assume an application as written in ROM creates a data object D by performing steps A and then C. In effect, A+C=D. After issuing smart cards with this application, the issuer identifies an error in the application in relation to the creation of data object D. The remedy for this error requires that D be created by steps A, then B, and then C, i.e., A+B+C=D. The conventional filtering approach would first create an "incorrect" D according to the ROM based application steps, and thereafter delete the incorrect data object D, and then re-create it using the EEPROM based application steps identified in the filtering instructions.

One can readily see that the filtering approach is very inefficient, and for more than one or two minor fixes results in a tangled mess of code. Not surprisingly, while this inefficient filtering approach to correcting a ROM based application works well enough for smart cards running a single application, it would not work for a smart card running multiple applications from different vendors.

The historic inability to securely modify or upgrade existing smart card applications in the field is just one reason why smart cards have failed to realize their full commercial potential. Many other reasons exist. Prominent among theses reasons is the absence of a true operating system (OS) supporting applications from multiple vendors.

A true operating system does not execute commands received from the outside world. Thus, in the context of a smart card, a true operating system will not (is unable to) execute commands received from a terminal. Rather, an operating system serves as a conduit and router for commands communicated from a terminal to an application stored on the smart card. Additionally, an operating system serves as a conduit through which an application utilizes the hardware resources. In other words, an operating system provides I/O functions and provides other functionality to applications running on the OS. Since first generation smart cards store only the application code, and since this code must necessarily executes commands received from the terminal, first generation smart cards do not include an operating system.

In an attempt to overcome the difficulties, limitations and expense associated with the programing of first generation smart cards, second generation smart cards incorporate an interpreter. An interpreter can be thought of as a library of commands. JAVA and BASIC are common interpreters. A set of commands is defined and identified in the interpreter, any one of which may be "called" by an application. The term "call" or "calling" is used throughout to broadly describe a relationship between two pieces of code in which one piece invokes the other. Commands, functions, definitions and instructions may be used by having one piece of code call another pieces of code. The foregoing pieces of code may reside within the an application or the OS.

Conceptually, an interpreter **33** can be thought of residing between application **30** and hardware resources **32**, as shown in FIG. 3B. Thus, an application running on a second generation smart card gains access to the hardware resources only through the interpreter which converts a command into one or more microprocessor instructions.

The interpreter effectively provides a higher level of abstraction and a programming language reflecting this level of abstraction with which a broader class of programmers may effectively write applications. However, the definition of commands by the interpreter, which promotes programming efficiency and standardization, necessarily restricts programing flexibility, since an interpreter will never define the entire range of commands theoretically made possible by an unrestricted combination of the microprocessor instructions. Thus, by use of an interpreter, programming flexibility is traded away for programming ease and standardization. The use of an interpreter also slows program execution since commands must be converted into microprocessor instructions before execution.

Further, since conventional smart cards implement the file structure defined by ISO-7816, part 4, the use of an interpreter comes as an additional penalty to programming flexibility. That is, ISO-7816, part 4 already confines an application programmer to a certain command set used to define a standard file architecture. On top of this restriction, the interpreter further confines the programmer to another

fixed set of commands. If a particular functionality is not defined by a command in the interpreter's library, the functionality can not be implemented within an application.

For example, if some new smart card application desired the function of outputting data to a printer, this function could not be implemented if the interpreter lacked the necessary command, such as a "PRINT" command commonly associated with desktop computers. Such functional inabilities attributable to the interpreter are particularly exasperating where a sequence of microprocessor instructions might be designed to implement the desired "new" function, but where the programmer lacks access to the microprocessor instruction set because of the obligatory presence of the interpreter.

Like the embedded application of first generation smart cards, both the application and the interpreter of second generation smart cards are stored in ROM. The process of correcting bugs is thus complicated by the possibility of fixing code in one or both of the interpreter and the application.

Even where modification or upgrading to a fielded smart card application can be accommodated in the conventional smart card, the physical process of reprogramming must be performed in a "trusted" device. That is, a terminal controlled by the smart card issuer must used to reprogram the smart card application in order to ensure code integrity. This is true because conventional smart cards, in and of themselves, do not have the processing capability or sophistication to verify new programming code.

Rather, conventional smart cards merely store security keys which are exported to the trusted machine during the security routine used to authenticate the new code. Thus, the trusted machine, not the smart card, performs the security routine. Such a process requires that the smart card export proprietary security keys. Once exported, the smart card loses control over the keys, thereby creating a number of additional opportunities for hackers to breach the card's security.

SUMMARY OF THE INVENTION

The present invention provides a system and method by which one or more applications may be securely installed on a smart card. Additionally, the present invention provides a system and method by which one or more applications may be securely de-installed from a smart card. The capabilities of a true operating system running on the smart card are used to facilitate the install and de-install operations.

In one aspect, the present invention allows a clear definition to be drawn between manufacture of the smart card and it's programming. Thus, a method of manufacturing a smart card is provided, where the smart card comprises a microprocessor and a memory, the memory comprising ROM and read/write memory. The method comprises storing at least a portion of an operating system (OS) in ROM, storing at least a boot application in ROM, and following complete manufacture of the smart card, providing the smart card to an issuer without material data stored in read/write memory. The material data comprises either a file structure or a security key.

In another aspect, the present invention provides a method of initializing a smart card. Assuming the smart card comprises a microprocessor and a memory and the memory comprises a ROM and read/write memory, the ROM storing an operating system (OS) and an application, the method comprises; powering the smart card via a terminal, determining whether the smart card has ever previously been powered, and upon determining that the smart card has never previously been powered, performing an first-power initialization routine.

The first-power initialization routine comprises initializing the read/write memory, locating the application in ROM, and installing the application in read/write memory. The application may comprise a boot application allowing insertion of a new data object into read/write memory, and a call to a download manager in the OS. Following installation of the application in read/write memory, an initialization routine may be performed which comprises building a memory management record, and sending an Answer-To-Reset (ATR) signal to the terminal. The initialization routine might further comprise; determining whether a transaction record is present in memory, and upon determining that a transaction record is present in memory calling a transaction manager, and by operation of the transaction manager, clearing the transaction record.

Given the potential length of the first-power initialization routine and the initialization routine, before performing either the smart card may send a first portion of the ATR signal to the terminal, and wherein the step of sending the ATR signal to the terminal following the step of building the memory management record comprises sending a second portion of the ATR signal to the terminal.

In another aspect of the present invention, the ROM-based boot application is installed upon first-power initialization and used to download a new application data object which is ultimately converted to a new application. To accomplish this, the boot application will return a command table when installed, the commands in the table allowing for the download of the new application data object and calling of a download manager. When called the download manager will in turn call the new application data object. The new application data object will create a command table for the new application which is stored in read/write memory. Before calling the download manager, the boot application may perform a security verification routine on the new application data object.

In yet another aspect, the present invention provides a method of downloading a second application from a terminal into a smart card memory, the memory storing an operating system (OS) and a first application, the method comprising; calling the first application from the terminal, by operation of the first application, inserting a new application data object in memory, and converting the new application data object into the second application. Before converting the new application data object into the second application, the first application may verify the authenticity of the new application data object using a digital signature, for example.

Within this method, the step of converting the new application data object into the second application comprises; calling a download manager in the OS, calling the second application from the download manager, and generating a command table for the second application.

Command tables and other persistent data records within the context of the present invention may be stored and accessed in a file directory using an efficient data record structure using OS based file manager.

In still another aspect, the present invention provides a method of de-installing an application on a smart card, the smart card comprising a microprocessor and a memory, the memory storing an operating system (OS) and the application, the method comprising; deleting all data objects associated with the application, and deleting all data records associated with the application.

In a related aspect, step of deleting all data objects associated with the application, and deleting all data records associated with the application comprises; beginning with a 1^{st} data object associated with the application and continuing through the N^{th} data object associated with the application, for each data object, deleting the data object by operation of the application, and calling the file manager, and by operation of the file manager, deleting the data record associated with the data object. Following deletion of the N^{th} data object and the N^{th} data record associated with the application, the method may thereafter delete the application data record from memory, and by operation of the memory manager reallocate memory space associated with the application as being available.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 shows an exemplary smart card;

FIG. 2 shows the integrated circuit portion of the exemplary smart card of FIG. 1 in some additional detail;

FIG. 3A illustrates the software/hardware relationship of a conventional, first generation smart card;

FIG. 3B illustrates the software/hardware relationship of a conventional, second generation smart card;

FIG. 4 is a flowchart illustrating a portion of an exemplary start-up routine including a first initialization routine and an initialization routine;

FIG. 5 is a flowchart illustrating a portion of an exemplary start-up routine including an ATR return;

FIG. 6 is a flowchart illustrating an exemplary routine by which a new application is downloaded;

FIG. 7 is a flowchart illustrating and exemplary routine to accomplish an install event, within the download routine of FIG. 6;

FIG. 8 illustrates an exemplary structure for a data record used within the present invention;

FIG. 9 conceptually illustrates a portion of read/write memory to further describe an install event; and

FIG. 10 is a flowchart illustrating an exemplary routine by which a new application is de-installed.

DETAILED DESCRIPTION

Some limitations inherent in conventional smart cards are manifest during the manufacturing process. As presently required, conventional smart cards pass from the smart card manufacturer to the smart card issuer with data necessarily stored in nonvolatile, read/write memory, as well as in ROM. In the explanation which follows, some arbitrary lines have been drawn between stages in an exemplary process by which smart cards are placed into commercial use. Such descriptive lines are intended merely to facilitate the explanation which follows. To begin, a "manufacturer" creates the physical smart card structure. There are relatively few smart card manufacturers in the world, and many present and potential smart card issuers.

As introduced in the foregoing section example, the term "smart card" was described with reference to the common commercial device shown in FIG. 1. While this example serves well for the explanations which follow, it should be noted that the present invention is broadly applicable to class of devices having physical form factors which are different from the one illustrated in the example. For example, the present invention is readily adapted to Secure Interface Modules (SIMs) and Secure Access Modules (SAMs). SIMs and SAMs are physically "cut-down" versions of the classic smart card, and are used with telephones and other spaces smaller than that typically provided by larger terminals. Obviously, the size, shape, nature, and composition of the material encapsulating or mounting the microprocessor and memory element are not relevant or limiting to the present invention. Thus, as used in the description that follows and as claimed below, the term "smart card" should be broadly read as encompassing any self-contained combination of microprocessor and memory element capable of performing a transaction with another device, generally referred to as a terminal.

The term "issuer" denotes the party responsible for at least one application on the smart card. That is, the issuer is the party ultimately responsible for the development, implementation, security, and/or utility of an application running on the smart card. The manufacturer and the issuer may be the same party, but are more likely than not different commercial entities. An issuer may sell or distribute smart cards directly or through various middlemen until they reach the ultimate customer, the "user."

The fact that a manufacturer will produce smart cards for more than one issuer presents some real concerns, given the conventional approach to smart card development. Efficiency requires that the manufacturer generate or mask certain code into ROM. Unfortunately, the manufacturer is also typically required to place certain data in read/write memory. Such data includes, for example, initial file structure information and specialized transport keys which are necessary to preclude fraudulent use of blank smart cards.

This requirement that the manufacturer place certain data in read/write memory inherently threatens an issuer's ability to control access to, and definition of the application. By handing a manufacturer proprietary keys, an issuer creates an entire range of potential security problems. Since many manufacturers are also issuers, or produce smart cards for other issuers, the potential for conflicts of interest is very real.

The present invention avoids these problems by providing a smart card which may be delivered from the manufacturer to an issuer without any data stored in read/write memory. The issuer may thus completely define an application without sharing proprietary information with the manufacturer.

In one aspect, the present invention accomplishes this result by providing a smart card having a true operating system (OS), as defined above. For purposes of explanation, the OS is said to implement or define a number of functions. As used in this capacity, the term "function(s)" should not be read as denoting only classic functionally programming operations, such as "MOVE," "STORE," "LOOP," etc., but should be read as including any operation or data state definable within the OS. Typically, an OS function is defined by a collection or sequence of instructions. However, an OS function might consist of only a single microprocessor instruction, or an OS function might be nothing more than a particular variable definition. Thus, the term function(s), as used throughout, should be construed as something having relation to or a definition within the OS.

In one aspect of the present invention, the OS is used in conjunction with an

Application Programing Interface (API) and/or a Hardware Application Layer (HAL). These features are described in a commonly assigned U.S. patent application Ser. No. 09/386,290 filed Aug. 31, 1999. The subject matter of this application is incorporated herein by reference.

When present, the API further defines, or indexes, the OS functions, or more preferably a subset of the OS functions.

In effect, the API forms a library of favorite, or most commonly used functions callable by applications on the smart card. In the common vernacular of programmers, the API is "glue" between the applications and OS. Thus, as a command is executed by an application, one or more OS functions may be called through the API. Conceptually, the API may be thought of as a vector table in which OS functions are immediately cross-referenced with an address in memory to the code implementing the function. Any application written for the smart card may make use of the API, but unlike the conventional smart card employing an interpreter, the application is not required to operate within the set of functions indexed within the API. The API, thus, provides an efficiency resource defining certain standard functions, without constraining an application to the use of only the standard functions.

An API allows shared data objects to be defined which may be referenced by any application running on the smart card. The API index of functions provides a reference for programmers during development of an application. As such, standardization and consistency of use are obtained. Further, the API provides an additional level of programming abstraction above the OS. By consistent recourse to the API, applications programmers may literally ignore the specific nature of the hardware resources accessed through the OS, and may also ignore much of the OS. This level of abstraction allows applications to be written in terms of a standard API which are able to run on literally any microprocessor.

The term "command" is used throughout to denote a data or control transaction between a terminal and a smart card application, or between one application and another. As described above, because a smart card according to the present invention uses a true OS, the OS can not execute a command. Rather, the OS facilitates transmission of the command from a terminal (or another application) to the application. When executed in the application, the command may result in one or more calls to the OS for certain functions. One or more of these calls may be made through the API, or may be made directly to the OS for functions which are not indexed within the API.

However, such function calls from the application to the OS, whether through the API or not, are optional—not mandatory. Within one aspect of the present invention, a smart card application may be stored in native form. That is, the application may, in whole or in part, be complied down to machine executable object code capable of running directly on the hardware resources of the smart card without any call to the OS.

The advantages of this option are profound. For example, assume in year one that a competent OS defining an adequate plurality of functions is commercially implemented on large number of smart cards. Further, assume a very popular application "X" runs on smart cards using this OS.

Next, assume in year two that a bright smart card applications programmer determines to implement an entirely new function within application X. The new function may be readily implemented using the existing microprocessor instruction set, albeit in a previously unheard of sequence, but has not been defined within the OS.

In this situation, the conventional first generation smart card would require a significant rewrite of the application. Further, since the first generation smart cards run on a fixed, embedded version of application X, all existing smart cards must be replaced with new cards in order to implement an upgraded version of the application X including the new function.

Conventional second generation smart cards can not implement the new function because the existing interpreter does not define it. Thus, in order to upgrade application X in second generation smart cards, a new version of the interpreter must be written and downloaded onto each card. Since second generation smart cards store the interpreter in ROM, and generally lack a mechanism to perform field downloads, particularly in relation to an interpreter, such cards must be physically replaced in the field in order to provide customers with the upgraded version of application X.

In the context of a smart card system in which code may be quickly and easily downloaded in the field, the option of having an upgraded application X bypass the OS to directly execute the new function in the microprocessor is particularly profound. For example, the party owning and/or administering application X may not be the same party owning the OS, and/or the smart card. Thus, the owner of application X may lack the ability or authorization to amend the OS. Similarly, the new function may be highly proprietary and as such the owner of application X may not wish to disclose the function to the OS or smart card provider.

In any event, application X may be upgraded in the field to incorporate code, which when directly executed on the microprocessor, provides the new function without call to the OS. Such limited amendment of application X leaves all other applications and the OS untouched. That is, the OS need not be modified to implement the new function. As a result, the relationship between the OS and all other applications stored on the smart card is not altered.

Alternatively, the OS might be upgraded to define the new function, and thereafter the API might be amended to index the new function. The process by which, in another aspect, the present invention amends, or replaces ROM-based programs such as the OS, API, and HAL or ROM based applications is referred to as "patching." Patching is described in a commonly assigned U.S. Pat. No. 6,338,435. The subject matter of this application is incorporated herein by reference.

In one embodiment of the present invention, the OS and one or more applications are stored in ROM. Additional applications may be stored in read/write memory, but at least the initial bulk of the programs will be stored in ROM. This embodiment is presently preferred on a cost basis, but it is recognized that as the performance characteristics and relative cost of various memory types, i.e., as between ROM, EEPROM, and their substitutes like Flash, change over time this preference may change.

Because the present invention provides a smart card having a true OS, the smart card memory must store at least a "boot" application. That is, because a true OS is unable to execute any command received from a terminal, some application must be resident on the smart card to receive an execute at least one beginning command. Assuming a conventional relationship in which a manufacturer is asked to mask an application(s) into ROM, and define read/write memory in relation to the application(s), a smart card according to the present invention operates normally, as explained below. However, the present invention provides much more.

Assume as one example that an issuer purchases a "minimal" smart card from a manufacturer. A minimal smart card, which when physically produced and delivered to the issuer for programming, includes the typical hardware resources, i.e., a microprocessor and a memory. The memory will store the OS and the boot application in ROM. Read/write memory, typically an EEPROM, will be empty.

At this point, it is further assumed that the OS defines a plurality of functions including a "download manager." The download manager is a piece of code which is capable of taking a data object stored in memory and converting it into a working application on the smart card. The data object thus converted is called an "application data object" to distinguish it from other types of data objects stored in memory.

The download manager often cooperates with a file manager and a memory manager. These two managers, as presently preferred, are described in a commonly assigned U.S. patent application Ser. No. 09/386,286 filed Aug. 31, 1999. The subject matter of this application is incorporated herein by reference.

The boot application stored in ROM must be capable of executing at least two commands, or their equivalent: (1) inserting a data object in read/write memory, and (2) calling the download manager. This statement is particularly true for the current example of a minimal smart card delivered to the issuer without anything loaded in read/write memory. In other words, the boot application accomplishes at least one task—downloading a second application. This second application may have any form, and may even be a new or more sophisticated boot application. Before performing this task, however, the boot application must, like all other applications running on a smart card according to the present invention, must be "installed."

Before explaining the install routine, an exemplary start-up routine will be described. This start-up routine, which may include many other features and functions not described here, will be explained with reference to the flowchart shown in FIG. 4.

When power is received in the smart card, it determines whether or not power has ever been previously applied to the smart card (40). This determination may be made, for example, by examining a predetermined memory location in read/write memory. If the smart card determines that this is the first time power has been applied (40=yes), it enters a "first-power initialization routine." As described below, a smart card according to the present invention will preferably run an "initialization routine" upon each power up. The first-power initialization routine is a specialized version of the initialization routine which is run one time following manufacture of the smart card. The first-power initialization routine may be run at the manufacturer's facility, at the issuer's facility, or at a terminal in the field when the smart card is first used. Both the first-power initialization routine and the initialization routine are preferably part of the OS, but may be included as part of one or more smart card applications.

With reference to FIG. 4, the first-power initialization routine comprises; defining the smart card starting environment (41), locating an application in memory (42), and installing the application (43). The definition of the smart card starting environment will very from card to card, but as presently preferred will at least establish a portion of read/write memory for storing global data objects, define a file directory in read/write memory, and initialize other variable, data, and file structures in memory. In effect, defining the smart card starting environment creates a place for subsequently installed application to live.

Once the starting environment has been defined, the boot application stored in ROM is located (42), and installed (43). This is the case in the present example which assumes that the only application stored in ROM is the boot application.

However, the present invention is also applicable to smart cards storing multiple applications in memory (i.e., ROM and/or read/write memory). Any application stored in memory might be installed during the first power initialization routine, but as explained above, an ability within the smart card to download additional applications requires that the application, whatever it other characteristics and wherever it is stored, be able to execute commands inserting a data object and calling the download manager.

With an application installed in read/write memory, the smart card ends the first power initialization routine and begins the initialization routine. Alternatively, as shown in FIG. 4, the initialization routine is begun upon a determination that the smart card has previously been powered (40=no).

The initialization routine may vary from card to card and from issuer to issuer, but as presently preferred the initialization routine comprises at least; building a memory management record (44), clearing a pending transaction, if any, (45), and sending ATR to the terminal indicating that the smart card is ready to begin a transaction (46). A preferred routine by which the memory management record is built is described in the. commonly assigned application referenced and incorporated above, U.S. patent application Ser. No. 09/386,286 filed Aug. 31, 1999. A preferred routine by which a pending transaction is described in commonly assigned U.S. patent application Ser. No. 09/386,287.

Another aspect of the present invention is described with reference to the flowchart in FIG. 5. As mentioned above, ISO-7816 defines the ATR signal used by conventional smart cards to initiate communication with a terminal. Given the relatively simple nature of conventional smart card operation, the existing ATR protocol does not necessarily contemplate a lengthy start-up routine including some or all of the foregoing. It is possible, therefore, for the terminal to prematurely terminate a session when an ATR signal is not received within an anticipated period of time.

The conventional ATR signal comprises multiple bytes of data. The present invention uses this fact, to avoid "timing-out" during the Reset signal/ATR exchange routine between a terminal and a smart card. For example, looking a FIG. 5, upon receiving power and a reset signal, the smart card begins a start-up routine (51). Soon after beginning the start-up routine, the smart card returns one or more bytes of the ATR signal (52) to essentially hold the terminal while completing the start-up routine (53). Once the start-up routine is complete, the smart card sends the remaining portion of the ATR signal (54) before entering an I/O loop, or performing some other function.

As described above, the smart card must install an application as part of the first-power initialization routine. In fact, once the starting environment of the smart card has been established, additional applications may also be installed at any time. Installed applications may have many forms, and may perform many tasks. For example, an installed application might be completely new to the smart card, or it might be an upgrade to an existing application. A new application might be persistent in the smart card until de-installed, or might be a one-use application which runs once and de-installs itself immediately thereafter. Where an existing application is stored entirely in read/write memory the upgrade process simply downloads and installs the upgrade application, and deletes the existing application. Alternatively, the upgrade application might overwrite all or some of the existing application in read/write memory.

As noted above, where an application is written in ROM, wholly or in part, the process of upgrading, replacing, or amended such ROM-based applications is referred to as

patching. Thus, as used herein, the concept of a "patch" has meaning with regard to ROM-based code. Applications, the OS, and API are ready examples of typical ROM-based programs which may be patched. Patches are typically one-use applications which create read/write memory-based code. Once established patch code is viewed as just another type of application which, in turn, may be upgraded or replaced by a "new application."

As one of ordinary skill in the art will recognize from the foregoing, the term "application" as used herein it not limited to just self-contained programs, but encompasses pieces of code which form part of a present, future, or existing application, whether such application is stored in ROM or in read/write memory.

Thus, regardless of the nature of the downloaded application or its purpose within the smart card, it may be installed according to the present invention. One presently preferred method for installing an a "new" application is described in relation to the flowchart of FIG. 6. To begin, an existing application is called (61). The term "existing application" is used to denote any responsive piece of code on the smart card. An existing application may be stored in ROM or in read/write memory. It may be a boot application or any other type of application. The existing application then receives a command to insert a "new data object" (62). Once the new data object is stored in read/write memory, the existing application receives a command to make a "new application" (63). The process of making a new application from the new data object may include an authentication routine (64). Once the new data object has been authenticated, the existing application calls the download manager (65). The download manager operates on the new data object to create the new application. This routine is referred to as an "install event" (66), and will be described in more detail below.

Of note, the "call" to the existing application, and/or the commands to download the new data object and make the new application may come from a terminal, another application running on the smart card, or the OS.

Of further note, the new data object is authenticated by the existing application before the download manager is called to perform the install event. Thus, while the OS may provide the resource necessary to install a new application, the existing application maintains the security mechanisms by which creation of the new application is allowed. This is an important feature in a system which allows multiple applications from multiple vendors to be run on the same smart card using a common operating system. Each issuer thus retains complete security control over the downloading of upgrade and replacement applications. The security mechanism need not be left to the OS and/or shared with the smart card manufacturer.

Further, this ability to authenticate new programming code "on card" allows a smart card according to the present invention to be updated, or to have a new application added in the field through a non-secure transmission medium like the Internet. As explained above, conventional smart cards can only facilitate upgrades, if at all, on trusted machines.

Thus, assuming for the moment that a conventional smart card has the ability to uprade an existing application in the field, such upgrading could only be done via a trusted machine since the smart card itself could not perform the authentication routine required to verify the new code. This is true for any new code downloaded to the conventional smart card.

In the exemplary method described above, the new data object need not be immediately installed. Rather, it may be left inertly residing in read/write memory until needed, if ever. Thus, a rarely used but marketable option to an application might be stored in read/write memory as a data object, without the overhead of a fully installed application segment, until required, if ever, by a user. Once needed, it may be installed and run as part of the application.

The install event described above is further described in relation to the flowchart shown in FIG. 7. The install event is controlled by the download manager, which is preferably part of the OS. When called by an application, or within the OS, to create a new application the download manager is always called in relation to a new data object, i.e., a data object stored in memory but not yet formed into a new application. At times, depending on the nature of hardware resources and the contents of memory, the download manager may be forced to relocate the new data object in read/write memory (71) prior to making the new application. Once properly situated in memory, the new data object is called by the download manager (72). In the context of the download manager, the new data object is really a new application data object as compared with other types of data objects stored in memory. Once called by the download manager, the new application data object will return a command table (73). Following return of the command table, the download manager will call the file manager (74) which will define within the file directory a data record associated with the command table (75). Having previously established a data record in the file directory for the new application data object, this data record must be updated by cooperation of the download manager and the file manager to reflect the conversion of the new application data object into a new application (76).

The command table may have various structures or be implemented in any number of different algorithms. In whatever form, the command table serves to identify every command executable by the new application. When a command is received in the OS via an I/O loop, for example, the smart card OS will seek to identify an application "owning" the command, i.e., an application able to execute the command. This is done by reference to the one or more command tables stored in memory. Like all persistent data objects and files in memory, each command table must be referenced in a file directory.

In addition to the command table, the new application may create any number of additional data objects associated with the new application. Such additional data objects may be data, variables, files, records, sub-applications or anything else required by the new application. Any additional data object intended to be persistent in memory will have an associated data record stored in the file directory.

FIG. 8 shows an exemplary structure for a data record. In one aspect of the present invention, each data record in the file directory is stored with a known structure or format. Thus, a command table data record will have the same basic structure as all other data records in the file directory. Data records within a file directory may be distinguished, for example, by their type field. While not required as part of the present invention, use of a standard data record structure provides notable benefits. As one of ordinary skill will understand, the exact nature, size, and characteristics of this "standard" data record structure can be left to the individual programmer. The explanation which follows is merely a presently preferred example.

FIG. 8 illustrates an exemplary 16-byte data record structure comprising a 2-byte ID field, a 1-byte ownership field, a 1-byte type field, a 4-byte data field, a 2-byte data length

field, and a 6-byte label field. As presently preferred the type field and the label field are user definable. That is, an application's programmer may use these data record fields for any purpose whatsoever. The download manager, file manager, memory manager, and the OS in general, do not demand that these fields be defined. They are merely variable data fields associated with a data record. As examples, the type field might indicate that the associated data object is an application, a command table, a file, or some other type data object. The label field might indicate the access type for the data record.

The ID field identifies the data record within the file system administered by the OS. The ownership field includes ownership information. In the present example, the ownership field of the data record contains the unique ownership byte previously described. Only the OS may access and define the ID and ownership fields in each data record.

The data field and the data length field are related within each data record. The data length field specifies the size of the data field. In the preferred embodiment, the data field is allocated 4 bytes, it's maximum data size. Thus, if the data length field indicates that the data is 4 bytes or less in size, then the data field stores the actual data associated with the data record. If, however, the data length field indicates that the data field is greater than 4 bytes in size, the data field stores a 4-byte data pointer indicating the beginning address, elsewhere in read/write memory at which the actual data may be found.

An example of an install event will now be described with reference to FIG. 9. A new application data object **90** has already been stored in read/write memory **99**. When called by the download manager, the new application data object returns at least a command table **92** which is stored in read/write memory. By operation of the file manager, a command table data record **95** is created in the file directory **98**. Other data objects, A and B, may also be created in read/write memory as part of the install event. If such optional data objects are intended to be persistent with read/write memory, each must define a corresponding data record in the file directory. After successfully creating the new application with all of its associated data objects and data records, the download manager and the file manager must change the nature of the original data record **91** associated with the new application data object.

For example, when stored in read/write memory, the new application data object data record indicated ownership by the "existing application." In other words, the existing application (the boot application in the minimal smart card example) which inserted the new application data object into read/write memory and called the download manager "owned" the new application data object at that time. Once converted into a new application the ownership field for the new application data record indicates ownership by the OS. That is, henceforth, the OS "own" the new application and deal with it as with all other previously installed applications. Similarly, the type field which once indicated a "data" type for the new application data object, is changed to an "application" type once the new application has been installed. With its data record properly defined in the file directory and having a defined command table, the new application is ready to receive commands from the OS I/O loop, for example.

By their very size smart cards include a limited set of resources. Memory space is perhaps the most critical smart card resource. Thus, any attempt to expand the capabilities of conventional smart cards to run multiple applications on a true operating system must address the issue of useful memory space conservation. The commonly assigned and previously incorporated U.S. patent application Ser. No. 09/386,286 discloses a system and method for reallocating read/write memory space. To facilitate the return and reuse of memory space, one aspect of the present invention provides a method of de-installing an application.

A presently preferred method for de-installing an application on a smart card will be explained with reference to the flowchart shown in FIG. **10**. The illustrated de-install event begins when the smart card receives a de-install application command in the OS from a terminal (**101**). The OS locates the application "owning" this command (**102**). Alternatively, a de-install event may arise from operation of an application on the smart card. That is, during operation, a first application may require de-installation of a second application or even de-installation of itself.

However initiated, the application executing the de-install event calls the download manager (**103**). The download manager calls the application to be de-installed (**104**). The application executing the de-install event may de-install itself or de-install some other application. Naturally, the ability of a first application to delete a second application requires the first (executing) application to "own" the second (de-installed) application. Such ownership is readily indicated by the ownership field of associated data record for the second application, for example.

The application to be deleted first deletes all associated data objects (**105**). The process of deleting data objects typically includes removing the data record associated each data object from the file directory. Again, all associated data objects may be readily recognized in memory by scanning through the file directory searching for data records having particular ownership field information. Once all associated data objects have been deleted, the file manager is called (**106**). The file manager will then delete the file directory entry for the data record associated with the de-installed application (**107**). Once the application has thus been de-installed, the memory manager is called (**108**), and memory space once allocated to the now de-installed application is reallocated as being "available" (**109**).

The long term commercial success of smart cards requires an ability to upgrade capabilities while the smart card remains in the user's possession. That is, smart cards must be able to receive new applications and upgrade existing applications in the field, preferably through non-secure media such as the Internet. Any number of conventional authentication routines may be used within the context of the present invention to verify data objects and commands. The OS of the present invention facilitates any reasonable security routine which may be implemented by a specific application. No one "OS-based" security model need be mandated.

The present invention provides a system and method for securely downloading new applications and existing application upgrades. A true smart card operating system is the only efficient way to accomplish secure implementation and operation of multiple applications from different vendors on a single card. Further, as smart cards begin to accept multiple applications from different vendors, the card's ability to install and de-install applications must operate without impacting the smart card operating system or other existing applications. Since there are relatively few smart card manufacturers which provide cards for all issuers, each smart card issuer will require a system which allows them complete control over the programming of the card.

The present invention has been described using several examples. The context needed to explain the download manager, initialization routine and install/de-install events includes some assumptions regarding data record use and storage, data record structure, data object organization in memory, and the type of memory commonly used. These contextual aspects are used to effectively explain the broader inventive concepts of the present invention. However, the present invention is not limited to the disclosed examples and teaching context. Rather the present invention is defined by the attached claims.

What is claimed:

1. A method of initializing a smart card, the smart card comprising a microprocessor and a memory, the memory comprising ROM and read/write memory, the ROM storing an operating system (OS) and an application, the method comprising:

powering the smart card via a terminal;

determining whether the smart card has ever previously been powered;

upon determining that the smart card has never previously been powered, performing an first-power initialization routine.

2. The method of claim 1, wherein the first-power initialization routine comprises:

initializing the read/write memory;

locating the application in ROM; and

installing the application in read/write memory.

3. The method of claim 2, wherein the application comprises a boot application allowing insertion of a new data object into read/write memory, and a call to a download manager in the OS.

4. The method of claim 3, further comprising:

following installation of the application in read/write memory, performing an initialization routine, comprising:

building a memory management record; and

sending an Answer-To-Reset (ATR) signal to the terminal.

5. The method of claim 4, further comprising:

before performing the first-power initialization routine, sending a first portion of the ATR signal to the terminal; and

wherein the step of sending the ATR signal to the terminal following the step of building the memory management record comprises sending a second portion of the ATR signal to the terminal.

6. The method of claim 1, wherein upon determining that the smart card has previously been powered, performing an initialization routine, comprising:

building a memory management record; and

sending an Answer-To-Reset (ATR) signal to the terminal.

7. The method of claim 6, wherein the initialization routine further comprises:

determining whether a transaction record is present in memory; and

upon determining that a transaction record is present in memory calling a transaction manager, and by operation of the transaction manager, clearing the transaction record.

8. The method of claim 6, wherein the initialization routine further comprises, after sending the ATR, entering an Input/Output routine.

9. The method of claim 6, further comprising:

before performing the initialization routine, sending a first portion of the ATR signal to the terminal; and

wherein the step of sending the ATR signal to the terminal following the step of building the memory management record comprises sending a second portion of the ATR signal to the terminal.

10. The method of claim 2, wherein installing the application in read/write memory comprises:

calling the application from the terminal;

receiving in the application a command to create a command table in read/write memory.

11. The method of claim 10, further comprising:

following creation of the command table in read/write memory, creating additional data objects in read/write memory.

12. The method of claim 3, wherein installing the application in read/write memory comprises:

calling the boot application from the terminal;

receiving in the boot application a first command, and in response thereto generating at least one data record and storing the at least one data record in read/write memory.

13. The method of claim 12, wherein the at least one data record is a command table for the boot application.

14. The method of claim 13, wherein following storage of the boot application command table, and in responsive to a second command from the terminal, downloading a new application from the terminal.

15. The method of claim 14, wherein the step of downloading the new application comprises:

inserting a new application data object in read/write memory, and converting the new application data object into a new application.

16. The method of claim 15, wherein the step of converting the new application data object into a new application comprises:

calling a download manager;

from the download manager, calling the new application data object;

by operation of the of the new application data object, creating a command table for the new application; and

storing the command table in read/write memory.

17. The method of claim 16, further comprising:

before calling the download manager, performing in the boot application a security verification routine on the new application data object.

18. A method of downloading a second application from a terminal into a smart card memory, the memory storing an operating system (OS) and a first application, the method comprising:

calling the first application from the terminal;

by operation of the first application, inserting a new application data object in memory; and

converting the new application data object into the second application.

19. The method of claim 18, further comprising:

before converting the new application data object into the second application, verifying the authenticity of the new application data object within the first application.

20. The method of claim 19, wherein verifying the authenticity of the new application data object comprises:

verifying a digital signature associated with the new application data object.

21. The method of claim 18, wherein the step of converting the new application data object into the second application comprises:

calling a download manager in the OS;

calling the second application from the download manager, and generating a command table for the second application.

22. The method of claim 21, further comprising:

following generation of the command table, calling a file manager in the OS; and

by operation of the file manager, generating a first data record associated with the command table and storing the first data record in memory.

23. The method of claim 22, wherein the step of inserting the new application data object comprises:

calling the file manager in the OS;

by operation of the file manager generating a second data record associated with the new application data object and storing the second data record in memory.

24. The method of claim 23, wherein the first and second data records each comprise an ownership field and a type field.

25. The method of claim 24, wherein the step of generating the second data record associated with the new application data object comprises:

defining the ownership field in the second data record to indicate first application ownership; and

defining the type field in the second data record to indicate a data object type.

26. The method of claim 25, wherein the step of generating the first data record associated with the new application comprises:

defining the ownership field in the first data record to indicate OS ownership; and

defining the type field in the first data record to indicate an application type.

27. The method of claim 23, wherein the command table comprises an index of all commands executable by the second application.

28. A method of de-installing an application on a smart card, the smart card comprising a microprocessor and a memory, the memory storing an operating system (OS) and the application, the method comprising:

receiving a de-install command;

locating the application owning the de-install command, and by operation of the owning application calling a download manager located in the operating system;

by operation of the download manager, deleting all data objects associated with the application; and

deleting all data record associated with the application.

29. The method of claim 27, wherein the step of deleting all data objects associated with the application, and deleting all data record associated with the application comprises:

beginning with a 1 st data object associated with the application and continuing through the Nth data object associated with the application, for each data object;

deleting the data object by operation of the application; and

calling the file manager, and by operation of the file manager, deleting the data record associated with the data object.

30. The method of claim 29, wherein an application data record associated with the application is stored in memory, the method further comprising:

following deletion of the Nth data object and the Nth data record associated with the application, deleting the application data record from memory.

31. The method of claim 30, further comprising:

following deletion of the application data record from memory, calling a memory manager and by operation of the memory manager reallocating memory space associated with the application as being available.

* * * * *